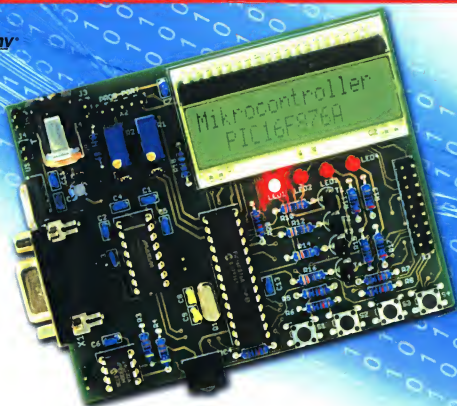


Электроника



Михазль Хофманн

МИКРОКОНТРОЛЛЕРЫ для начинающих

На компакт-диске:

- приведены примеры программ
- чертеж для изготовления монтажной платы
- электрические схемы, техническая документация
- справочная информация и программное обеспечение

FRANZIS

Michael Hofmann

Mikrocontroller für Einsteiger

Schaltungen entwerfen und Software programmieren

Mit 102 Abbildungen

Mikrocontroller für Einsteiger
Michael Hofmann

Михаэль Хофманн

МИКРОКОНТРОЛЛЕРЫ для начинающих

Санкт-Петербург

«БХВ-Петербург»

2010

УДК 621.382
ББК 32.85
Х86

Хофманн М.

Х86 Микроконтроллеры для начинающих: Пер. с нем. — СПб.: БХВ-Петербург, 2010. — 304 с.: ил. + CD-ROM — (Электроника)

ISBN 978-5-9775-0551-2

Рассмотрено программирование микроконтроллеров на примере PIC16F876A компании Microchip. Подробно описаны основные команды языка ассемблер, а также среда разработки MPLAB. Показано программирование с помощью отладчика-программатора ICD 2, а также через последовательный интерфейс. На практических примерах рассмотрено управление светодиодами и дисплеем, представление аналоговых сигналов в цифровой форме, сохранение/запись данных во внешнюю EEPROM-память, управление выходами микроконтроллера с помощью ИК-пульта дистанционного управления и др. На компакт-диске приведены примеры программ, чертеж для изготовления монтажной платы, электрические схемы, техническая документация, справочная информация и программное обеспечение.

Для радиолобителей

УДК 621.382
ББК 32.85

Группа подготовки издания:

Главный редактор	<i>Екатерина Кондукова</i>
Зам. главного редактора	<i>Игорь Шишигин</i>
Зав. редакцией	<i>Григорий Добин</i>
Перевод с немецкого	<i>Виктора Букирева</i>
Редактор	<i>Юрий Рожко</i>
Компьютерная верстка	<i>Ольги Сергиенко</i>
Корректор	<i>Зинаида Дмитриева</i>
Оформление обложки	<i>Елены Беляевой</i>
Зав. производством	<i>Николай Тверских</i>

Die berechnete Übersetzung von deutschsprachiges Buch Mikrocontroller für Einsteiger, ISBN: 978-3-7723-4318-6. Copyright © 2009 Franzis Verlag GmbH, 85586 Poing. Alle Rechte vorbehalten, auch die der fotomechanischen Wiedergabe und der Speicherung in elektronischen Medien. Das Erstellen und Verbreiten von Kopien auf Papier, auf Datenträger oder im Internet, insbesondere als PDF, ist nur mit ausdrücklicher Genehmigung des Verlags gestattet und wird widrigenfalls strafrechtlich verfolgt. Die Russische Übersetzung ist von BHV St. Petersburg verbreitet, Copyright © 2010.

Авторизованный перевод немецкой редакции книги Mikrocontroller für Einsteiger, ISBN: 978-3-7723-4318-6. Copyright © 2009 Franzis Verlag GmbH, 85586 Poing. Все права защищены, включая любые виды копирования, в том числе фотомеханического, а также хранение и тиражирование на электронных носителях. Изготовление и распространение копий на бумаге, электронных носителях данных и публикации в Интернете, особенно в формате PDF, возможны только при наличии письменного согласия Издательства Franzis. Нарушение этого условия преследуется в уголовном порядке. Перевод на русский язык "БХВ-Петербург" © 2010.

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 31.05.10.

Формат 70×100/16. Печать офсетная. Усл. печ. л. 24,51.

Тираж 2000 экз. Заказ № 273

"БХВ-Петербург", 190005, Санкт-Петербург, Измайловский пр., 29.

Санитарно-эпидемиологическое заключение на продукцию
№ 77.99.60.9533.Д.005770.05.09 от 26.05.2009 г. выдано Федеральной службой
по надзору в сфере защиты прав потребителей и благополучия человека.

Отпечатано с готовых диапозитивов

в ГУП "Типография "Наука"

199034, Санкт-Петербург, 9 линия, 12

ISBN 978-3-7723-4318-6 (нем.)
ISBN 978-5-9775-0551-2 (рус.)

© 2009 Franzis Verlag GmbH, 85586 Poing
© Перевод на русский язык "БХВ-Петербург", 2010

Оглавление

Предисловие	1
1. Обзор микроконтроллеров	3
1.1. Сравнительные характеристики микроконтроллеров.....	6
1.2. Структура и принцип работы PIC16F876A.....	6
1.2.1. Блок-схема.....	6
1.2.2. Флэш-память программ.....	8
1.2.3. Обработка данных в АЛУ.....	9
1.2.4. Регистр состояния.....	10
1.2.5. Адресация оперативной памяти или регистров ОЗУ.....	10
1.2.6. Вызов подпрограмм.....	12
1.2.7. Косвенная адресация.....	14
1.2.8. Чтение и запись внутренней EEPROM-памяти.....	16
2. Команды ассемблера PIC16F876A	21
2.1. Обзор команд.....	22
2.2. Подробное описание команд ассемблера.....	24
2.2.1. Общее.....	25
2.2.2. Форматы чисел.....	26
2.2.2.1. Двоичный формат.....	26
2.2.2.2. Восьмеричный формат.....	27
2.2.2.3. Шестнадцатеричный формат.....	27
2.2.2.4. Десятичный формат.....	27
2.2.2.5. ASCII-формат.....	28
2.2.2.6. Подведение итогов.....	29
2.2.3. Логические операции.....	30
2.2.4. Команды сдвига.....	38
2.2.5. Арифметические команды.....	43
2.2.6. Команды передачи управления.....	47
2.2.7. Прочие команды.....	58
3. Программирование с помощью MPLAB	63
3.1. Установка MPLAB.....	64
3.2. Настройка каталога проекта.....	64
3.3. Создание проекта.....	65
3.4. Рабочий стол MPLAB.....	69

3.5. Меню <i>View</i>	74
3.5.1. Аппаратный стек.....	75
3.5.2. Окно наблюдения.....	75
3.5.3. Листинг дизассемблера.....	76
3.5.4. EEPROM-память.....	77
3.6. Точки останова.....	77
3.7. Симулятор.....	78
3.7.1. Основные настройки.....	79
3.7.2. Асинхронный стимул.....	79
3.7.3. Циклический синхронный стимул.....	80
3.7.4. Другие вкладки окна <i>Stimulus</i>	82
3.8. Логический анализатор.....	82
3.9. Внутрисхемный отладчик ICD 2.....	84
3.10. Программирование.....	91
3.11. Текстовый редактор.....	92
4. Программный интерфейс.....	95
4.1. Программирование с помощью ICD 2.....	95
4.2. Процесс программирования.....	98
4.3. Биты конфигурации.....	99
4.3.1. Генератор.....	100
4.3.2. Сторожевой таймер.....	101
4.3.3. Таймер включения питания.....	102
4.3.4. Обнаружение провала напряжения.....	102
4.3.5. Низковольтное программирование.....	103
4.3.6. Защита чтения данных из EEPROM-памяти.....	103
4.3.7. Запись Flash-памяти программы.....	103
4.3.8. Защита кода.....	104
4.3.9. Обзор битов конфигурации.....	104
4.4. Микроконтроллеры OTP-типа.....	105
5. Монтажная плата.....	107
5.1. Описание схемы аппаратных средств.....	107
5.1.1. Блок питания.....	108
5.1.2. Интерфейс программирования.....	108
5.1.3. Генерация тактовых импульсов.....	109
5.1.4. Задание аналоговых напряжений.....	109
5.1.5. Кнопки.....	110
5.1.6. Индикация выходных сигналов на светодиодах.....	111
5.1.7. Приемник инфракрасного излучения.....	112
5.1.8. EEPROM-память.....	112
5.1.9. Интерфейс RS-232.....	113
5.1.10. Жидкокристаллический индикатор.....	113
5.1.11. Разъем для расширения.....	114

5.2. Программное обеспечение	115
5.2.1. Подключение внешних файлов.....	115
5.2.2. Биты конфигурации	116
5.2.3. Определения	116
5.2.4. Переменные.....	117
5.2.5. Макрокоманды	117
5.2.6. Начало программы.....	118
5.2.7. Инициализация.....	119
6. Входы и выходы	121
6.1. Расположение выводов PIC16F876A.....	121
6.2. Обзор функций выводов.....	123
6.3. Цифровые входы и выходы.....	126
6.4. Пример программы "Управление светодиодами".....	130
7. Таймер	133
7.1. 8-разрядный таймер (Timer0).....	134
7.2. 16-разрядный таймер (Timer1).....	135
7.3. Модуль таймера Timer2.....	141
8. Обработка аналоговых сигналов.....	145
8.1. Аналого-цифровое преобразование.....	145
8.1.1. АЦП-преобразование методом поразрядного уравнивания.....	147
8.1.2. Передаточная функция АЦП.....	150
8.1.3. Вычисление значения напряжения.....	151
8.1.4. Выравнивание оцифрованного значения.....	152
8.2. Пример программы "Вольтметр"	153
8.3. 16-битное сложение	156
8.4. 16-битное вычитание	157
8.5. Анализ оцифрованного значения.....	157
9. Отображение данных на индикаторе	163
9.1. Контроллер индикатора.....	163
9.1.1. Набор символов	164
9.1.2. Способы управления индикатором	166
9.2. Инициализация индикатора	168
9.3. Интерфейс аппаратных средств.....	170
9.3.1. Подпрограмма для передачи команды.....	171
9.3.2. Подпрограмма для передачи символа.....	173
9.3.3. Макрокоманда для инициализации индикатора.....	174
9.4. Пример программы "Hello World".....	175
10. Отображение на индикаторе аналогового напряжения	179
10.1. Вычисление напряжения	179
10.2. Подпрограмма "AD_konvertieren"	181

10.3. Преобразование двоичного числа в десятичное число	184
10.4. Основная программа.....	187
11. Измерение мощности и сопротивления.....	191
11.1. Измерение тока	191
11.2. Двоичное умножение.....	192
11.3. Двоичное деление	196
11.4. Отображение расчетной мощности.....	201
11.5. Отображение рассчитанного сопротивления.....	205
12. Передача данных посредством последовательного интерфейса	213
12.1. Последовательный интерфейс RS-232	214
12.1.1. Подключение через последовательный интерфейс	214
12.1.2. Протокол интерфейса RS-232	215
12.2. Программное обеспечение для передачи данных	217
12.3. Применение интерфейса USART.....	218
12.3.1. Установка скорости в бодах.....	219
12.3.2. Установка регистров TXSTA и RCSTA	220
12.4. Пример программы "Управление с помощью компьютера".....	221
13. Передача данных по шине I²C	227
13.1. Принцип работы интерфейса I ² C	227
13.2. Управление памятью EEPROM	229
13.3. Пример программы "Сохранение измеренных значений в EEPROM-памяти"	232
13.3.1. Подпрограмма <i>Schreibe_EEPROM</i>	236
13.3.2. Подпрограмма <i>Lese_EEPROM</i>	238
14. Переключение с помощью инфракрасного дистанционного управления.....	245
14.1. Протокол RC5.....	246
14.2. Пример программы "Инфракрасный переключатель".....	250
Приложение.....	259
Распределение в памяти регистров микроконтроллера PIC16F876A.....	259
Обзор регистров управления и состояния.....	260
Регистр состояния — STATUS.....	261
Регистр опций — OPTION_REG	262
Регистр контроля прерываний — INTCON	263
Первый регистр прерывания от периферии — PIR1	264
Второй регистр прерывания от периферии — PIR2	265
Регистр разрешения периферийных прерываний — PIE1	266
Регистр разрешения периферийных прерываний — PIE2	267
Регистр контроля питания — PCON	268

Регистр управления модулем таймера 1 — T1CON.....	269
Регистр управления модулем таймера 2 — T2CON.....	270
Регистр состояния модуля MSSP — SSPSTAT (режим SPI)	271
Регистр состояния модуля MSSP — SSPSTAT (в режим I ² C)	272
Регистр управления модулем MSSP — SSPCON (режим SPI)	274
Регистр управления модуля MSSP — SSPCON (режим I ² C)	275
Второй регистр управления модулем MSSP — SSPCON2 (режим I ² C).....	276
Регистр управления модулем Сравнения/Захвата/ШИМ — CCPxCON.....	277
Регистр состояния и управления приемника модуля USART — RCSTA	278
Регистр состояния и управления передатчика модуля USART — TXSTA.....	280
Регистр управления модулем АЦП — ADCON0	281
Регистр управления модулем АЦП — ADCON1	282
Регистр управления модулем компаратора — CMCON	283
Регистр управления опорным напряжением компаратора — CVRCON.....	284
Регистр управления косвенной записи/чтения EEPROM-памяти данных и Flash-памяти программ — EECON1	285
Список источников информации	286
Описание компакт-диска	287
Предметный указатель	291

Предисловие

Микроконтроллеры находятся почти во всех электронных устройствах. В этой книге показано, что запрограммировать и применить микроконтроллер не так уж сложно. Хотя у некоторых при упоминании слова "Ассемблер" (Assembler) в связи с программированием "волосы встают дыбом". После прочтения этой книги вы поймете, что это вовсе не так уж сложно, как это иногда кажется.

Возможности микроконтроллера рассмотрены на различных примерах, в которых используется микроконтроллер *PIC16F876A* производства компании Microchip, имеющий различные интерфейсы и обладающий достаточно широкими возможностями. На примерах будет показано, как осуществить опрос входов и переключение выходов микросхемы. Вы также научитесь управлять дисплеем для отображения текста и данных. Узнаете, как измерить аналоговые сигналы, сохранить их в *EEPROM*-памяти (Electrically Erasable Programmable Read-Only Memory — электрически стираемое программируемое постоянное запоминающее устройство) и затем прочитать с помощью персонального компьютера. На другом примере будет показано управление выходами с помощью инфракрасного управления. Часть примеров может моделироваться с помощью среды разработки *MPLAB*, поэтому в принципе не потребуются какие-либо аппаратные средства. Но поскольку любой микроконтроллер должен когда-нибудь хоть раз использоваться для схемы, представляется маленькая монтажная плата, на которой все примеры могут испытываться в реальной среде. Для облегчения работы и отладки монтажа вы можете приобрести неуконфигурированную (без деталей) печатную плату, если посетите мой сайт (www.edmh.de). Если вы захотите сами дополнить ее, то найдете расположение схемных элементов в Eagle-формате на приложенном CD-ROM. На нем находится также различная документация, например, обзор команд и описания регистров микроконтроллера. Для программирования микроконтроллера используется внутрисхемный отладчик *ICD 2* (In-Circuit Debugger) от компании Microchip. Однако на CD-ROM находится очень простая схема, с которой PIC-контроллер может программироваться через последовательный интерфейс.

Я желаю вам большого удовольствия и успеха при программировании микроконтроллера. Я всегда открыт для критики, похвалы и рационализаторских предложений и рад корреспонденции, поступающей на адрес моей электронной почты info@edmh.de.

Михаэль Хофманн

1. Обзор микроконтроллеров

Сегодня микроконтроллеры имеются почти в каждом электронном устройстве. Они применяются в термометрах для отображения температуры и управляют требуемым составом кофе в кофеварках. Микроконтроллеры автоматически открывают и закрывают ворота гаражей и выручают шофера в опасных ситуациях с помощью противоблокировочных устройств (ABS) и системы антивибрационной защиты (ESP).

Микроконтроллер (сокращенно МК, англ. MC) или также *микрокомпьютер* — это электронная микросхема, которая может осуществлять вычисления и управление техническими объектами и технологическими процессами подобно персональному компьютеру. Разумеется, МК чаще предназначен для управления конкретным устройством, а персональный компьютер способен выполнять обработку всех данных и обычно управляет многими разными устройствами. Персональный компьютер более универсален, поскольку используется для обработки текста, для компьютерных игр и многого другого, а поэтому должен быть очень гибок и предоставлять в распоряжение пользователя достаточное пространство памяти и необходимую вычислительную мощность. Напротив, микроконтроллер имеет узконаправленное предназначение. Если он используется, например, только для управления температурой в помещении, тогда для вычислений не требуется высокая производительность и не нужен большой объем памяти. При этом должна измеряться и обрабатываться лишь температура. Разумеется, строгую границу между микроконтроллером и персональным компьютером провести нельзя. Микроконтроллер в мобильном телефоне необходим для выбора номера абонента и обеспечения телефонного разговора. Однако при помощи современных карманных компьютеров можно отправлять электронные письма, записывать видео и слушать музыку.

Поскольку обычно для каждого специального задания не изготавливают узкоспециализированный микроконтроллер, поэтому из достаточно большого количества моделей универсальных контроллеров можно вполне выбрать

подходящий. Имеются разные производители, которые изготавливают много вариантов контроллеров, которые едва ли принципиально отличаются режимами функционирования. Микроконтроллеры располагают разным количеством входов и выходов, а также специальными аппаратными модулями, находящимися внутри, с которыми упрощается выполнение различных заданий. Таким образом, почти каждый микроконтроллер имеет таймер, с помощью которого можно устанавливать время и генерировать сигналы определенной длительности. Кроме того, многие микросхемы располагают встроенным *аналого-цифровым преобразователем* (АЦП), который позволяет измерять аналоговые сигналы для контроля, например, температуры или напряжения батареи.

Габариты микросхем определяются преимущественно количеством входов и выходов. Таким образом, микроконтроллер, который должен наблюдать только за напряжением батареи и выдавать сигнал при выходе за пределы порогового значения, может обходиться малым количеством выводов. Напротив, контроллер, который должен регулировать температуру в нескольких комнатах и отображать этот процесс управления на цветном индикаторе с сенсорным экраном, нуждается в существенно большем количестве входов и выводов. Поэтому у производителей можно найти микроконтроллеры как с шестью выводами, так и с несколькими сотнями выводов. Кроме того, можно заметить, что обычно с увеличением количества выводов микросхемы МК вычислительная мощность также возрастает, поскольку большее количество выводов позволяет решить соответственно и большее число разнообразных задач. Также количество битов, которые одновременно обрабатываются, растет с габаритами микроконтроллера. В настоящее время имеются модули с разрядностью шины от 4 до 32 битов. Для персонального компьютера уже применяются 64-битные процессоры. В скором времени микроконтроллер тоже наверняка сможет оперировать 64-битными значениями. Будет ли это теперь преимуществом или скорее недостатком, каждый разработчик должен решать сам. Микроконтроллеры с длиной слова 4 бита можно найти преимущественно в музеях и очень старых устройствах. Однако они продаются еще и сегодня для восстановления старых устройств. В современных разработках эти микросхемы больше не используются. В настоящее время больше всего распространены микроконтроллеры с длиной слова от 8 до 16 битов. Выбор среди них очень велик. А вот 32-битные микроконтроллеры применяются преимущественно для устройств, которые управляют цветным графическим индикатором (дисплеем), поддерживают различные носители информации, такие как USB- или SD-карты, и предназначены для подключения к персональному компьютеру для обмена данными...

В большинстве случаев 8-битных микроконтроллеров бывает вполне достаточно и к тому же их приобретение менее затратно. Кроме того, небольшие

электронные схемы могут быть реализованы без изготовления печатной платы. Так как многоразрядные микроконтроллеры функционируют практически аналогично контроллерам с меньшим числом разрядов, то в дальнейшем преимущественно будут приведены схемы с использованием 8-битного микроконтроллера. Представленные схемы и примеры программ достаточно просто реализовать, смоделировать и испытать.

В дальнейшем рассматриваются только контроллеры фирмы *Microchip*. Все примеры были запрограммированы для PIC-микроконтроллеров (*PIC* — *Programmable Integrated Circuit* — программируемая интегральная схема) этой фирмы. Контроллеры других производителей, например *Atmel*, *Motorola*, *Renesas* и т. д., функционируют по такому же принципу и отличаются преимущественно возможностями и системой команд.

Почему выбор в этой книге пал на продукты *Microchip*, зависело, в том числе, оттого, что фирмой *Microchip* предоставляется в распоряжение пользователей полностью бесплатная среда разработки. Контроллеры можно купить у известных поставщиков электронных комплектующих изделий и они, обладая большой популярностью, имеют много подробных примеров и обсуждений в Интернете.

Представленные в этой книге примеры программировались и испытывались при помощи встроенного отладчика *In-Circuit-Debugger (ICD2)* от компании *Microchip*. Поэтому все рисунки и описания относятся к этому программатору. Однако имеются также другие программаторы разных производителей, при помощи которых можно программировать и отлаживать микроконтроллер. Также в Интернете находится много инструкций для изготовления самодельных программаторов. Руководство по разработке простого PIC-программатора имеется на приложенном CD-ROM. Но он подходит только для программирования контроллера, отладка (поиск ошибок) с этим простым программным адаптером не возможна.

Программирование осуществлялось на языке программирования ассемблер. В этом случае лучше усваивается принцип работы микроконтроллера, оптимально используются все его системные ресурсы. Также в Интернете можно найти несколько бесплатных компиляторов для языка программирования C. На прилагаемом компакт-диске имеется полный, рабочий программный код с комментариями к каждому примеру. Но чтобы гарантировать оптимальное обучение, требуется пытаться самостоятельно программировать примеры и только в крайнем случае использовать версии, находящиеся на компакт-диске. Программы не оптимизированы на самое короткое время выполнения или самый короткий программный код. Они должны лишь демонстрировать многостороннюю функциональность микроконтроллера.

Все примеры программ в этой книге представлены для микроконтроллера PIC16F876A. У этой микросхемы есть преимущество в том, что она имеет все

основные характерные элементы современных микроконтроллеров. Количество контактов ввода/вывода I/O этого контроллера вполне достаточно для реализации практически любого проекта, однако при необходимости возможен очень простой переход и на другие типы. При этом нужно обязательно проверять названия и адреса регистров, поскольку не все PIC-контроллеры используют одни и те же адреса и обозначения.

1.1. Сравнительные характеристики микроконтроллеров

Следующий краткий обзор представляет только маленькую часть имеющихся в настоящее время микроконтроллеров. Перечень только поясняет, насколько различны микроконтроллеры:

- ☐ *PIC10F222* — очень маленький микроконтроллер с шестью выводами и размером 2×3 мм);
- ☐ *PIC16F876* — 8-битный микроконтроллер с 28 выводами;
- ☐ *PIC24FJ128* — 16-битный микроконтроллер с 64—100 выводами;
- ☐ *PIC32MX460* — 32-битный контроллер с 64—100 выводами;
- ☐ *ARM9* — 32-разрядный микроконтроллер с ядром ARM (Advanced RISC Machines) в корпусе *BGA* (Ball Grid Array — конструкция корпуса микросхемы с выводами в виде крошечных металлических шариков. Они расположены в виде сетки на его нижней поверхности, которые прижимаются к контактным площадкам на печатной плате без применения пайки. Преимущество — более низкая стоимость изготовления и уменьшение размеров при наличии более трехсот выводов).

1.2. Структура и принцип работы PIC16F876A

1.2.1. Блок-схема

Для обеспечения работы микроконтроллера требуется много маленьких функциональных блоков, которые должны оптимально взаимодействовать. На рис. 1.1 показана упрощенная функциональная схема микроконтроллера PIC16F876A.

Самой важной составной частью микроконтроллера является арифметикологическое устройство — *АЛУ* (англ. *ALU* — Arithmetic Logic Unit). С помощью АЛУ, которое является *операционным устройством* контроллера,

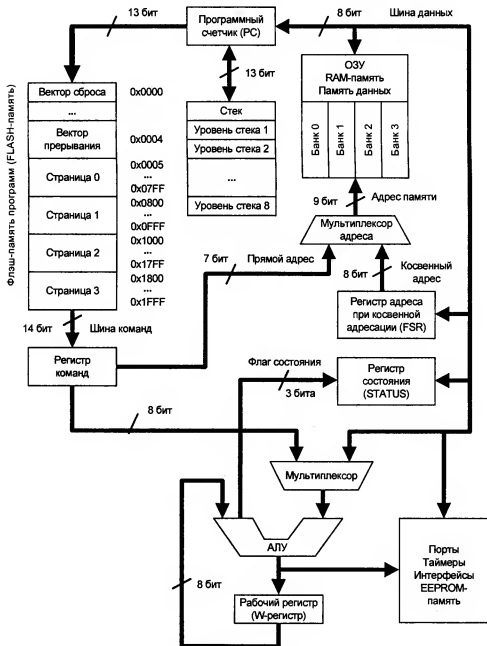


Рис. 1.1. Структурная схема PIC16F876A

выполняются арифметические и логические операции. Чтобы АЛУ могло оперировать с соответствующими значениями, они должны предоставляться в распоряжение в определенное время из соответствующей области памяти.

1.2.2. Флэш-память программ

Сначала разберемся, как данные поступают в микроконтроллер и как они образуются. Микроконтроллер не понимает, к сожалению, такой язык программирования, как Basic, C или ассемблер, а воспринимает только двоичные значения, т. е. нули или единицы. Они записываются во *флэш-память программ* (англ. Flash Program Memory), начиная с адреса 0x0000. Поскольку Flash-память — это энергонезависимая память, в которой данные сохраняются и после выключения питания, то именно здесь и хранят программный код микроконтроллера. Если бы программу поместили в оперативное запоминающее устройство (ОЗУ) (англ. RAM), то устройство перестало бы функционировать после выключения. Поэтому ОЗУ это память, которая предназначена для хранения промежуточных данных, а не программ. Чтобы сформировать цифровые данные, которые понимает PIC-контроллер, необходимо выполнить несколько последовательных операций. Принципиально можно было бы написать программный код непосредственно в двоичном виде и загрузить его при помощи программатора в микроконтроллер. Однако любой разработчик понимает, что это очень трудоемкая работа, и может восприниматься как "штрафная". Поэтому для программирования микроконтроллеров, как правило, используют различные языки программирования. Наиболее употребительным является язык C или ассемблер. Оба языка очень близки к аппаратным средствам и поэтому могут непосредственно обмениваться сообщениями с физически имеющимися регистрами. Для сложных многоразрядных микроконтроллеров почти всегда применяется язык C, а вот относительно простой контроллер, который рассматривается в этой книге, достаточно часто программируют на ассемблере. Следующий пример показывает программу обратного счета от 9 до 0, написанную на языке C, ассемблере и в машинном коде, который в итоге должен находиться в PIC-контроллере.

Язык C	Ассемблер	Машинный код
for(i=9; i>0; i--);	movlw 0x09	000C 3009
	movwf 0x20	000D 00A0
	counter	000E 0BA0
	decfsz 0x20, 1	000F 280E
	goto counter	

Как можно видеть на представленном примере, было бы слишком непонятно и затруднительно программировать микроконтроллер непосредственно в машинных кодах. Поэтому применяют специальную программу, которая транслирует программу, написанную на языке С или ассемблере, в машинный язык. Называют эту программу *компилятор* (англ. *compiler*). Компилятор проходит шаг за шагом программный код и интерпретирует запрограммированные разработчиком команды. Это часто выполняется не за один, а за несколько проходов. После компилирования отдельные модули еще должны связываться между собой с помощью другой специальной программы — *компоновщика* (англ. *linker*). Если все правильно работает и ошибки отсутствуют, то после этого получают машинный код, который может быть применен принципиально только на предусмотренном контроллере. Во время процесса компиляции генерируется файл с расширением *hex*, в котором записано, в каком месте должны находиться в памяти отдельные команды. Далее этот файл может быть загружен при помощи программатора в микроконтроллер.

Теперь программа записана во флэш-память программ. Если просмотреть вышеупомянутый машинный код, то можно определить место в программном коде по первым четырем шестнадцатеричным цифрам. Код начинался бы в памяти с адреса `0x000C` и завершался бы на адресе `0x000F`. Теперь для выполнения программы должна определяться исходная точка. Это — адрес `0x0000`, с которого выполняется программа после запуска (*Start*) или после сброса (*Reset*). С помощью внешнего или внутреннего тактирования *программный счетчик* PC (англ. *Program Counter*) (или иначе *счетчик команд*) будет постепенно прибавлять 1 и запускать очередную команду после предыдущей. При отсутствии команд перехода и циклов в программе процесс продолжался бы максимально до адреса `0x1FFF` (что соответствует 8292 командам) и повторялся бы после этого опять сначала. Счетчик команд имеет разрядность 13 битов ($2^{13} = 8192$) и может обращаться поэтому максимально к 8292 командам. Команда имеет разрядность 14 битов (например, `movlw 0x09 = 0x3009 = 11 0000 0000 10012`). После обработки команды из регистра команд данные и адреса ОЗУ через внутренние линии передаются соответственно в АЛУ или оперативную память (ОЗУ).

1.2.3. Обработка данных в АЛУ

Данные обрабатываются после поступления в АЛУ. Команда `movlw 0x09` означает: "Загрузить шестнадцатеричное значение `0x09` в рабочий регистр W". Без использования рабочего регистра W в PIC-контроллере практически ничего не работает. Через него должны проходить почти все значения. Например, чтобы записать то или иное значение в ячейку оперативной памяти или иначе в тот или иной регистр ОЗУ (*File Register* — сокращенно регистр F),

оно должно быть сначала загружено в рабочий регистр W (`movlw 0x09`) и только после этого может передаваться дальше в регистр ОЗУ. Это может происходить, например, по команде `movwf 0x20`, которая означает не что иное, как: "Переслать значение из регистра W в регистр ОЗУ по адресу памяти 0x20".

1.2.4. Регистр состояния

Поскольку после выполнения некоторых команд необходимо знать, стал ли результат нулевым после выполнения математической или логической операции или же произошел перенос, в регистре состояния (STATUS) в зависимости от команды устанавливаются определенные биты. Речь идет о так называемых *флагах* или *битах состояния*. *Флаг* (Flag) — это признак или индикатор для определения того, что произошло после той или иной операции. В регистре состояния микроконтроллера имеются следующие флаги (рис. 1.2): C — флаг переноса (от англ. Carry), DC — флаг десятичного переноса (от англ. Digit Carry) и Z — флаг нулевого результата (от англ. Zero). Флаг переноса указывает на перенос из старшего бита, если, например, после сложения результат больше не может представляться 8-ю битами. Флаг десятичного переноса похож на флаг переноса, но показывает перенос из младшего полубайта, т. е. из четвертого бита. И, наконец, флаг нулевого результата показывает, является ли результат нулем после выполнения арифметической или логической операции или нет.



Рис. 1.2. Формат регистра состояния

1.2.5. Адресация оперативной памяти или регистров ОЗУ

При обращении к регистрам ОЗУ имеется небольшая проблема. Адресация осуществляется с помощью 9-разрядного адреса оперативной памяти (RAM-

адреса). С помощью 9 битов можно обращаться максимум к $2^9 = 512$ байтам (0x000—0x1FF). Разумеется, при прямой адресации в PIC-контроллере используется только 7 битов, что позволяет обратиться лишь к $2^7 = 128$ байтам (0x00—0x7F). Для решения этой проблемы прибегли к следующему трюку: всю память данных (ОЗУ) разделили на 4 банка, при этом каждый банк состоит из 128 регистров, среди которых имеются *регистры общего назначения* (англ. *GPR* — General Purpose Registers) и *специального назначения* (англ. *SFR* — Special Function Registers). Таким образом, чтобы обратиться к конкретному регистру, вначале нужно выбрать банк, а затем уже в нем найти требуемый регистр. *Выбор банка памяти* осуществляется с помощью двух битов в регистре состояния RP0 и RP1 (рис. 1.3). С помощью этих двух дополнительных битов и получают 9-разрядный адрес оперативной памяти: младшие 7 разрядов определяются из регистра команд при прямой адресации, а два старших бита устанавливаются посредством регистра состояния (рис. 1.4). Здесь также может подстерегать еще одна проблема, которая может возникнуть при программировании: если во время программирования добавляются строки кода, которые обращаются к регистру другого банка, можно легко забыть о переключении банка. Программа в таком случае транслируется правильно, но может повести себя не так, как требуется. Особенно эта опасность возникает, если реализуется выбор банков памяти в пределах *макрокоманды* (инструкция, директива как элемент типичного языка программирования, при обработке разворачивающаяся в определенную последовательность простых команд) или подпрограммы. Тогда после возврата к выполнению основной программы можно не заметить, что будет активен уже другой банк.

RP0 = 1	RP0 = 0	RP0 = 1	RP0 = 0
RP1 = 1	RP1 = 1	RP1 = 0	RP1 = 0
0x180	0x100	0x080	0x000
0x181	0x101	0x081	0x001
Банк 3	Банк 2	Банк 1	Банк 0
0x1F0	0x17E	0x0FE	0x07E
0x1FF	0x17F	0x0FF	0x07F

Рис. 1.3. Организация банков памяти

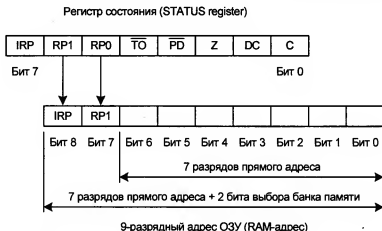


Рис. 1.4. Формирование RAM-адреса. Диапазон адресов банков памяти

1.2.6. Вызов подпрограмм

Схожее действие с макрокомандами выполняют и подпрограммы. Они служат для упрощения выполнения программы и пишутся для часто используемых последовательностей команд (например, "Передать данные через последовательный интерфейс"). Подпрограмма программируется только раз и поэтому занимает одни и те же ячейки в памяти программ. В этом случае подпрограмма может легко вызываться всякий раз, когда это необходимо. Чтобы вызвать подпрограмму, требуется небольшая процедура. Поскольку подпрограмма только раз сохраняется в памяти программ, но, как правило, неоднократно должна вызываться, обратиться к ячейке памяти начала подпрограммы можно с помощью программного счетчика. Чтобы это сделать, нужно выполнить команду перехода к подпрограмме или иначе — команду вызова подпрограммы — `call` (вызов) (рис. 1.5). При этом программный счетчик устанавливается на адрес начала подпрограммы и продолжает работу как обычно. После выполнения подпрограммы нужно вновь вернуться продолжить основную программу. Это происходит по команде `return` (возврат). Только какое значение нужно загрузить в программный счетчик? Для этого в PIC-контроллере при вызове подпрограммы значение программного счетчика временно сохраняется в стеке. *Стек* (англ. *stack*) — это память, организованная по принципу *LIFO* (англ. *Last In — First Out*), т. е. "последним пришел — первым вышел". Новое значение в стеке всегда помещается в верхнюю ячейку, которая называется *вершиной стека*. Вызывается это значение из стека опять-таки из вершины стека, причем в первую очередь. Аналогичным образом это функционирует со значением программного счетчика и при вызове подпрограммы. По команде `call` следующее за текущим значение про-

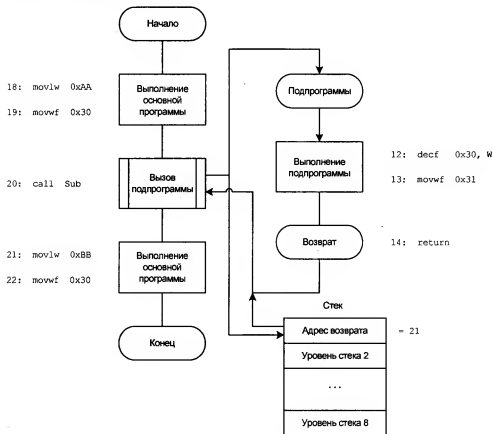


Рис. 1.5. Вызов подпрограммы

граммного счетчика помещается на вершину стека, а по команде `return` значение с вершины стека снова загружается в программный счетчик. Таким образом, работа основной программы может быть продолжена. Стековая память микроконтроллера PIC16F876A имеет глубину равную 8, т. е. она содержит 8 уровней. Другие уровни стека могут заполняться аналогично вершине стека, при условии вызова из данной подпрограммы очередной подпрограммы, которая в этом случае называется *вложенной*. Команда `call` в подпрограмме также помещает следующее за текущим значение программного счетчика на вершину стека (предыдущее значение вершины опускается ниже) и переходит на выполнение вложенной подпрограммы. Возврат к прерванной подпрограмме из вложенной осуществляется также по команде `return`. Таким образом, в данном микроконтроллере могут выполняться максимум 8 вызовов подпрограмм. После этого стековая память будет полностью заполнена, и больше сохранить адрес возврата будет невозможно.

1.2.7. Косвенная адресация

Иногда требуется рассчитывать адрес и обращаться к нему косвенно. "Косвенно" — значит не вводить непосредственно точный адрес, а указывать

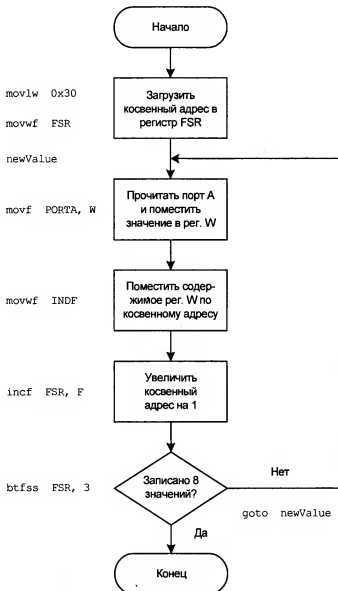


Рис. 1.6. Косвенная адресация

только ссылку на этот адрес. Для этих целей применяется только так называемый *указатель*. Этот метод может использоваться, например, чтобы последовательно записывать или очищать несколько регистров. Следующий пример показывает, как 8 значений по очереди считываются из входного порта А и заносятся в RAM-память.

В описанном примере данные считываются с входов порта А и заносятся в RAM-память по адресам от 0x30 до 0x37.

Так как при косвенной адресации для определения адреса имеются в распоряжении только 8 битов регистра FSR, то при этом можно обращаться далеко не ко всем регистрам RAM-памяти, которая, как известно, имеет 9-разрядный адрес. Поэтому при косвенной адресации снова используют регистр состояния (STATUS), а точнее один старший его бит IRP (рис. 1.7). Чтобы сделать выбор между банками 0—1 и 2—3, нужно установить или сбросить бит IRP. Так если бит IRP установить в 0, то можно обращаться к банкам 0 и 1. Если же бит IRP установить в 1, то адресуются банки 2 и 3. После этого содержимое 8-разрядного регистра FSR объединяется с битом IRP регистра состояния и таким образом образуется 9-разрядный адрес памяти данных. Для осуществления косвенной адресации используют *регистр косвенной адресации* INDF, адрес которого находится в каждом банке по адресу 0x00. Регистр в микроконтроллере физически отсутствует, он используется только как вспомогательное средство для адресации. Если пишут что-то в этот регистр, то запись осуществляется в физически имеющийся регистр, адрес которого находится в регистре FSR. Это же самое происходит и при чтении регистра INDF.



Рис. 1.7. Структура RAM-адреса для косвенной адресации

1.2.8. Чтение и запись внутренней EEPROM-памяти

Поскольку данные, которые сохраняются в регистрах RAM-памяти, пропадают после выключения микроконтроллера, часто необходимо, чтобы данные могли храниться продолжительно. При небольших объемах данных, например, при сохранении последнего рабочего состояния перед выключением, это может происходить через внутреннюю память *EEPROM* (Electronic Erasable Programmable Read Only Memory — электрически стираемое программируемое постоянное запоминающее устройство). В нее можно записывать данные, которые будут храниться так долго, сколько нужно. Запись и чтение памяти EEPROM, к сожалению, осуществляется не так просто, как в случае с регистрами ОЗУ. Судить об этом можно уже только потому, что имеются 6 регистров специального назначения для записи и чтения EEPROM- и Flash-памяти. В микроконтроллере PIC16F876A объем EEPROM-памяти данных равен 256 байтам. Если этого пространства памяти недостаточно, то можно, кроме того, задействовать неиспользованное пространство Flash-памяти программ. Разумеется, в этом случае нужно быть очень осторожным, если программа еще должна расширяться. Также может случиться, что при программировании не было обращено внимание на допущенные ошибки. Данные основной программы могут быть случайно стерты. В таком случае PIC-контроллер окажется неработоспособным, и должен будет вновь программироваться — на этот раз уже безошибочной программой.

Таким образом, лучшей альтернативой расширения EEPROM-памяти данных является добавление внешней EEPROM-памяти, в которую будут заноситься данные. Как подключить внешнюю EEPROM-память и управлять ею, объясняется далее в последующих главах. Исходя из этого, применение памяти программ для хранения данных должно быть тщательно продумано и учтены все возможные затраты. Так внешняя 32-килобитная EEPROM-память стоит примерно 0,40 € (≈17 руб.) за штуку. Поэтому в дальнейшем обсуждается только обмен данными с внутренней EEPROM-памятью. Разумеется, запись Flash-памяти программ предоставляет еще преимущество и в том, что программу при необходимости можно изменить. Таким образом, микропрограммное обеспечение, которое загружается через последовательный интерфейс, возможно совершенствовать.

Следующая блок-схема алгоритма (рис. 1.8) описывает процесс записи данных в EEPROM-память по определенному адресу. Операция записи в EEPROM-память по сравнению с оперативной памятью (RAM-памятью) длится гораздо дольше. Так для слова данных (одного байта) требуется примерно от 4 до 8 мс. При тактовом сигнале процессора 4 МГц это соответствует обработке 4000 команд. К счастью, процесс записи протекает в фоновом

```

_BANK_3
btfsc EECON1, WR
goto $-1

```

```

_BANK_2
movlw 0x24
movwf EEADR
movlw 0x11
movwf EEDATA

```

```

_BANK_3
bcf EECON1, EEPGD

```

```

bsf EECON1, WREN
bcf INTCON, GIE

```

```

movlw 0x55
movwf EECON2
movlw 0xAA
movwf EECON2
bsf EECON1, WR

```

```

bsf INTCON, GIE
bsf EECON1, WREN

```

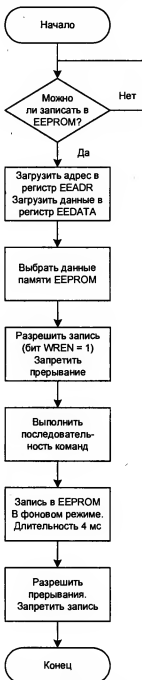


Рис. 1.8. Запись во внутреннюю EEPROM-память

режиме и не требуется ждать его окончания для продолжения работы основной программы. Длительность процесса записи также показывает, что нельзя использовать EEPROM-память для часто изменяющихся данных. Поскольку EEPROM-память подлежит старению, то имеется ограничение максимального количества записей в нее. Если производитель указывает количество минимум 100000 циклов стирания/записи, то это значит, что при сохранении каждую секунду нового значения в EEPROM-памяти, она будет непригодна уже примерно через 28 часов. Следует заметить, что для чтения EEPROM-памяти требуется то же самое время, как и для чтения при косвенной адресации.

Для записи EEPROM-памяти данных требуются только 4 из 6 специализированных регистров.

- ☐ EECON1 — регистр для управления записью в память EEPROM;
- ☐ EECON2 — регистр для запуска непосредственно записи;
- ☐ EEADR — регистр для хранения адреса для записи или чтения;
- ☐ EEDATA — регистр для данных, которые должны писаться в память EEPROM.

Регистры EEADRH и EEDATH дополнительно требуются для обращения к Flash-памяти программ. Это происходит потому, что для адресации к этой памяти требуется количество битов больше 8, т. е. обычной разрядности регистров специального назначения.

Как можно видеть при выполнении программы, чтобы выполнить процесс записи, нужно придерживаться специальной последовательности команд. Эта последовательность должна соблюдаться, т. к. иначе EEPROM-память не будет перезаписана. Также нужно обращать внимание на то, что во время выполнения этой последовательности прерывания микроконтроллера были отключены, поскольку иначе с передачей управления в программу обработки прерывания требуемая последовательность будет не соблюдена.

Чтение из EEPROM-памяти проще. На следующей блок-схеме алгоритма (рис. 1.9) представлены немногочисленные шаги, которые требуются для операции чтения.

Поскольку EEPROM-память данных это относительно медленная память, которая должна перезаписываться очень редко, то она хорошо подходит только для сохранения эталонных данных или для последовательностей значений. Поэтому объем равный 256 байтам для большинства применений также вполне достаточен.

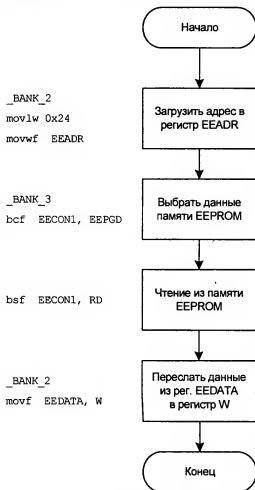


Рис. 1.9. Чтение из внутренней EEPROM-памяти

2. Команды ассемблера PIC16F876A

Микроконтроллер PIC16F876A — это микроконтроллер, построенный на основе процессора с RISC-архитектурой или иначе на основе *RISC-процессора* (*RISC* — Reduced Instruction Set Computer — процессор с сокращенным набором команд), т. е. контроллер обходится ограниченным набором команд. Так в распоряжении микроконтроллера PIC16F876A всего 35 команд ассемблера, причем среди них только простые логические и арифметические команды. Преимущество такой архитектуры — это быстрое выполнение отдельных команд. Все команды выполняются за один цикл, кроме инструкций переходов, которые выполняются за два цикла. Но учтите: длительность цикла команды не равна периоду тактовой частоты процессора. Цикл выполнения инструкции состоит из 4 периодов тактовой частоты. Таким образом, при тактовой частоте процессора, а точнее частоте кварцевого резонатора или генератора равной 4 МГц, время выполнения команды составляет 1 мкс, т. е. в секунду процессор может выполнить примерно 1 млн команд.

В противоположность RISC-процессору — процессор с архитектурой *CISC* (Complex Instruction Set Computer — это процессор со сложным набором команд) или иначе *CISC-процессор* имеет в своем распоряжении, как видно из названия, уже наиболее полный набор команд. При CISC-архитектуре команда состоит из нескольких маленьких микрокоманд. Преимущество таких архитектур в том, что само программирование из-за разнообразия команд заметно упрощается. Разумеется, при этом нужно знать большее количество команд, которые имеют, в зависимости от сложности, разную продолжительность выполнения, в большинстве случаев равную нескольким тактам. Чтобы получить такие же удобства при программировании с использованием RISC-процессора, нужно определять собственные команды и записать их в форме макрокоманд. *Макрокоманда* — это команда, вызывающая выполнение некоторой определенной последовательности других команд. Они пишутся перед трансляцией программы в месте, в котором вызывается макрокоманда.

Как конкретно функционируют макрокоманды и в чем заключены сильные и слабые их стороны, подробнее см. в *главе 5*.

Так как в АЛУ возможны только простые логические операции, все команды могут быть сформированы на основе логических операций. Содержимое регистров можно объединять с помощью логических операций И (AND), ИЛИ (OR) и "исключающее ИЛИ" (XOR). Для перемещения битов влево или вправо содержимое регистров можно соответствующим образом циклически сдвигать. Данные могут только складываться и вычитаться. Команд для умножения или деления не существует, а поэтому при необходимости выполнения этих операций их нужно сформировать из определенной последовательности основных команд, обращение к которым можно организовать с помощью команд перехода. Эти команды нужны, например, для организации циклов и вызовов подпрограмм, когда обязательно необходимо выполнение переходов на другой программный адрес.

2.1. Обзор команд

Обзор команд показан в табл. 2.1. В дальнейшем с ее помощью можно быстро найти информацию о любой рассмотренной в книге команде. Обзор команд для просмотра и вывода его на печать находится также на прилагаемом к книге компакт-диске.

Сначала приведем объяснение употребляемых в обзоре параметров.

- £ Указатель регистра обозначает адрес регистра в оперативной памяти; это может быть любой регистр в RAM-памяти. Однако т. к. он может адресоваться только с помощью 7 битов, нужно перед выполнением команды выбирать соответствующий банк.
- w Регистр W означает рабочий регистр или иначе аккумулятор.
- b Номер отдельного бита в 8-разрядном регистре.
- k С помощью k указывается константа или метка.
- d Указатель адресата результата операции. При помощи d указывается, где должен сохраняться результат операции. Если $d = 0$ — результат остается в регистре W, а если $d = 1$ — результат перемещается в регистр, указанный в £.

Таблица 2.1. Обзор команд

Команда	Параметр	Пояснение
Логические операции		
andwf	f, d	Побитная логическая операция И над содержимым регистров W и f
andlw	k	Побитная логическая операция И константы k и содержимого регистра W
iorwf	f, d	Побитная логическая операция ИЛИ над содержимым регистров W и f
iorlw	k	Побитная логическая операция ИЛИ константы k и содержимого регистра W
xorwf	f, d	Побитная логическая операция "исключающее ИЛИ" над содержимым регистров W и f
xorlw	k	Побитная логическая операция "исключающее ИЛИ" константы k и содержимого регистра W
clrf	f	Очистить содержимое регистра f
clrw	-	Очистить содержимое регистра W
comf	f, d	Инвертировать все биты регистра f
bcf	f, b	Очистить бит b в регистре f
bsf	f, b	Установить бит b в регистре f
Команда сдвига		
rlf	f, d	Циклический сдвиг влево содержимого регистра f через бит переноса C регистра состояния (STATUS)
rrf	f, d	Циклический сдвиг вправо содержимого регистра f через бит переноса C регистра состояния (STATUS)
movlw	k	Переслать константу k в регистр W
movwf	f	Переслать содержимое регистра W в регистр f
movf	f, d	Переслать содержимое регистра f в регистр адресата (регистр W, если d = 0, или регистр f, если d = 1)
swapf	f, d	Поменять местами старший и младший полубайты регистра f. Результат в регистре W, если d = 0, или регистре f, если d = 1
Арифметические команды		
addwf	f, d	Сложить содержимое регистров W и f. Результат в регистре W, если d = 0, или регистре f, если d = 1
addlw	k	Сложить содержимое регистра W с 8-разрядной константой k. Результат в регистре W
subwf	f, d	Вычесть содержимое регистра W из содержимого регистра f. Результат в регистре W, если d = 0, или в регистре f, если d = 1
sublw	k	Вычесть содержимое регистра W из 8-разрядной константы k. Результат сохраняется в регистре W

Таблица 2.1 (окончание)

Команда	Параметр	Пояснение
incf	f, d	Прибавить 1 к содержимому регистра f. Результат в регистре W, если d = 0, или в регистре f, если d = 1
decf	f, d	Вычесть 1 из содержимого регистра f. Результат в регистре W, если d = 0, или в регистре f, если d = 1
Команда передачи управления		
goto	k	Выполнить безусловный переход, т. е. переход к указанному адресу
call	k	Перейти на выполнение подпрограммы
return	-	Возвратиться из подпрограммы
retlw	k	Выполнить возврат из подпрограммы с загрузкой константы k в регистр W
retfie	-	Выполнить возврат из подпрограммы обработки прерываний и разрешить прерывания
incfsz	f, d	Прибавить 1 к содержимому регистра f. Результат в регистре W, если d = 0, или в регистре f, если d = 1. Если результат равен 0, то пропустить следующую команду
decfsz	f, d	Вычесть 1 из содержимого регистра f. Результат в регистре W, если d = 0, или в регистре f, если d = 1. Если результат равен 0, то пропустить следующую команду
btfss	f, b	Проверить бит b в регистре f. Пропустить следующую инструкцию, если бит b равен 1
btfsc	f, b	Проверить бит b в регистре f. Пропустить следующую инструкцию, если бит b = 0
Прочие		
nop	-	Нет операции (пустая операция)
clrwdt	-	Очистить сторожевой таймер WDT и предделитель, если он подключен к WDT
sleep	-	Перейти в режим сверхнизкого энергопотребления SLEEP

2.2. Подробное описание команд ассемблера

В табл. 2.1 представлен обзор имеющихся в распоряжении команд. Разумеется, для команд требуется еще нескольких объяснений, чтобы можно было по-

нимать их и правильно применять. Далее команды рассматриваются более подробно и поясняются на небольших примерах.

2.2.1. Общее

При программировании на ассемблере команды записываются друг под другом. Каждая команда стоит в новой строке. *Метка* (англ. label) может устанавливаться перед каждой командой. Она может использоваться как признак передачи управления. С применением меток также улучшается удобочитаемость кода. Метка может стоять перед каждой командой или находиться в собственной строке. Она всегда относится к следующей команде.

Пример использования меток:

```
movlw 0x0A      ;Загрузить в регистр W 0x0A (= 10)
movwf 0x20      ;Переместить содержимое регистра W в регистр ОЗУ 0x20
loop            ;Метка перехода
nop             ;Нет операции
decfsz 0x20     ;Уменьшать содержимое регистра 0x20 на 1, до тех пор
goto loop      ;пока значение не будет равно 0, конец цикла
```

Пример программного цикла с десятикратным повторением команды `nop` (нет операции). До тех пор пока содержимое регистра 0x20 не станет равно 0, программа снова и снова будет переходить к метке `loop`, которая стоит перед командой `nop`.

Как можно увидеть в примере программы, после каждой строки кода следует комментарий, который начинается с точки с запятой ";". Текст, который стоит правее точки с запятой, не будет при трансляции программы приниматься во внимание. Принципиально не нужно экономить в программах ассемблера на комментариях, даже и в том случае, когда вы полагаете, что это очень простая программа. При повторном обращении к программе по прошествии некоторого времени вы будете благодарны, что не поленились написать комментарий.

Во многих командах при помощи указателя адресата результата операции `d` (от англ. destination) можно указывать место сохранения результата. Его можно выбирать между регистром `W` или регистром ОЗУ, указанным с помощью `f`. Если `d = 0`, то результат после выполнения операции будет сохранен в рабочем регистре `W`. Если же `d = 1`, то в регистре, указанным в `f`. Поскольку вначале часто это путают, можно воспользоваться такой подсказкой:

W — **W**enig (перевод с нем. "мало"), т. е. когда `d = 0` — результат в регистре `W`;

F — **F**iel (Viel) (перевод с нем. "много"), т. е. когда `d = 1` — результат в регистре `f`.

Или же лучше применить такую подсказку:

Поскольку в команде "w" стоит перед "f" — wf, аналогично тому, как 0 по порядку предшествует 1, то соответственно при d = 0 результат будет в регистре W, а при d = 1 в регистре, указанном в f.

Пример:

```
andwf f, 0      ;Результат будет в регистре W
andwf f, 1      ;Результат будет в регистре, указанном в f
```

Чтобы обойти эту проблему, можно сделать шпаргалку, в которой W определяется как 0 и F как 1. Это же происходит в массиве данных, в котором регистры также определены.

```
W EQU H'0000'
F EQU H'0001'
```

Поэтому команды можно записать также следующим образом:

```
andwf f, W      ;Результат поступает в регистр W
andwf f, F      ;Результат поступает в регистр F
```

2.2.2. Форматы чисел

В цифровой технике имеются разные возможности для изображения чисел. В принципе, все двоичные числа состоят из нулей и единиц. Так как работа с двоичными числами во многих случаях затруднительна, то часто обращаются к другим форматам.

2.2.2.1. Двоичный формат

При использовании двоичного формата любое число представляется только двумя цифрами: 1 и 0. В соответствии с этим десятичное число 234_{10} имеет следующий вид в двоичном формате 11101010_2 . Поскольку ассемблер сам не может различить формат, в котором представлено то или иное число, например, числа 110 может быть представлено как в двоичном, так и десятичном формате, поэтому формат представления числа обязательно должен быть указан с помощью специальных символов. Например, число 110, если оно представлено в двоичном формате, имеет значение 6_{10} , а вот то же самое число, представленное в десятичном формате, это уже 110_{10} . Согласитесь, очень большая разница — 6_{10} или 110_{10} !

Чтобы указать, что число представлено в двоичном формате, слева от числа добавляют латинский символ "B" (от англ. Binary). Двоичное число устанавливается между одинарными апострофами.

$234_{10} \rightarrow B'11101010'$

Двоичный формат особенно хорошо подходит для установки или сброса отдельных битов. Можно посмотреть первоначально, какие биты были установлены в регистре, а какие нет. Применение для этой же цели числа 234, представленного в десятичном формате, менее наглядно и поэтому более сложно.

2.2.2.2. Восьмеричный формат

Для представления чисел в восьмеричном формате используются цифры от 0 до 7. Число в восьмеричном формате легко может быть преобразовано в двоичный формат. Для преобразования достаточно каждую цифру заменить *двоичной триадой*, состоящей из 3 битов.

Например, десятичное число 234_{10} представляется в восьмеричном формате как 352_8 . Представление числа в восьмеричном формате не так часто находит применение на практике. Чтобы сообщить ассемблеру, что число представлено в восьмеричном формате, надо слева от числа добавить символ "O" (от англ. Octal), а число заключить между одинарными апострофами.

$234_{10} \rightarrow \text{'O'352'}$

2.2.2.3. Шестнадцатеричный формат

Наиболее часто используемый формат — это шестнадцатеричный формат, в котором цифру можно представить двоичной тетрадой, состоящей из 4 битов. Поскольку с помощью 4 битов можно представлять 16 чисел, то цифры указываются цифрами от 0 до 9 и буквами от A до F. Шестнадцатеричный формат дает возможность очень короткой формы записи 8-битного значения. Требуется всего лишь 2 цифры. Причем форму записи можно выбирать между двумя видами представления. В первом случае, чтобы сообщить ассемблеру, что число представлено в шестнадцатеричном формате, надо слева от числа добавить символ "H" (от англ. Hexadecimal), а число заключить между одинарными апострофами. Однако в этой книге для шестнадцатеричных чисел используется второй вариант "0x...".

$234_{10} \rightarrow \text{'H'EA'}$ или

$234_{10} \rightarrow \text{0xEA}$

2.2.2.4. Десятичный формат

Для людей десятичный формат удобней и привычней всего. Здесь цифры представляются от 0 до 9. Ассемблер интерпретирует число как десятичное, если выбирается один из двух видов представления. В первом случае, чтобы сообщить ассемблеру, что число представлено в десятичном формате, надо

слева от числа добавить символ "D" (от англ. Decimal), а число заключить между одинарными апострофами. Во втором случае перед числом ставится только точка (которую можно легко перепутать с обычной десятичной точкой).

$234_{10} \rightarrow D'234'$ или

$234_{10} \rightarrow .234$

В этой книге для десятичного формата выбран первый вариант с буквой "D", поскольку это менее двусмысленно.

2.2.2.5. ASCII-формат

С помощью ASCII-формата можно представлять буквы. ASCII — это American Standard Code for Information Interchange (Американский стандартный код для обмена информацией). Для того чтобы закодировать тот или иной символ, в 8-битном коде ASCII имеется специальная таблица символов (табл. 2.2). Варианты этой таблицы кодов могут содержать символы для разных национальных языков. В таблице кроме информационных символов имеются и управляющие непечатаемые символы.

Ассемблер понимает символ как ASCII-код, если перед ним имеется символ "A", а далее следует сам информационный символ, заключенный между одинарными апострофами. В другом варианте можно обойтись даже без символа "A".

$M \rightarrow A'M' = 0x4D$ или

$M \rightarrow 'M' = 0x4D$

В приведенной таблице представлено только 128 символов, в то время как с помощью 8 битов можно закодировать 256 символов. Здесь показаны только первые 128 символов, т. к. они используются на всех языках. Для символов от 128 (0x80) до 255 (0xFF) речь идет о национальных и специальных символах.

Уже упомянутые управляющие символы в таблице имеют коды, начиная от 0x00 до 0x1F. Самые важные управляющие символы — это символы перевода строки LF (Line Feed) и возврата каретки CR (Carriage Return). В случае вывода текста при встрече символа LF курсор переходит на следующую строку, а когда встречается символ CR, курсор переводится в начало строки. При обработке текста функции этих обоих управляющих символов объединены в одной клавише <Enter>.

Таблица 2.2. Набор символов ASCII

Младший разряд (младший полубайт)	Старший разряд (старший полубайт)								
	HEX	0	1	2	3	4	5	6	7
	0	NUL	DLE	Space	0	@	P	.	p
	1	SOH	DC1	!	1	A	Q	a	q
	2	STX	DC2	"	2	B	R	b	r
	3	ETX	DC3	#	3	C	S	c	s
	4	EOT	DC4	\$	4	D	T	d	t
	5	ENQ	NAK	%	5	E	U	e	u
	6	ACK	SYN	&	6	F	V	f	v
	7	Bell	ETB	'	7	G	W	g	w
	8	BS	CAN	(8	H	X	h	x
	9	HAT	EM)	9	I	Y	i	y
	A	LF	SUB	*	:	J	Z	j	z
	B	VT	ESC	+	;	K	[k	{
	C	FF	FS	,	<	L	\	l	
	D	CR	GS	-	=	M]	m	}
	E	SO	RS	.	>	N	^	n	~
	F	SI	US	/	?	O	_	o	DEL

2.2.2.6. Подведение итогов

В среде проектирования MPLAB можно устанавливать, какой формат чисел должен выбираться, если никакие дополнительные символы ранее не устанавливались. Разумеется, нужно быть очень осторожным, чтобы быть уверенным, что выбран правильный формат чисел.

В завершение предоставляю еще одну небольшую таблицу с форматами чисел (табл. 2.3), которые используются в этой книге. Многоточие обозначает ту или иную последовательность цифр или символов.

Таблица 2.3. Форматы чисел

Формат	Синтаксис	Пример
Двоичный	B'...'	B'01101101'
Восьмеричный	O'...'	O'736'

Таблица 2.3 (окончание)

Формат	Синтаксис	Пример
Десятичный	D'...'	D'259'
Шестнадцатеричный	0x...	0x7D
ASCII	A'...'	A'j'

2.2.3. Логические операции

Команда: `andwf`

Назначение: Побитная логическая операция И (AND) над содержимым регистра W и f

Синтаксис: `andwf f, d`

Изменяемые флаги: Z

По команде `andwf` выполняется побитная логическая операция И над содержимым регистра W и регистром, адрес которого указан параметром f, т. е. F-регистром. Результат, в зависимости от параметра d, записывается в регистр W или F.

При d = 0 результат сохраняется в регистре W или, если d = 1 — в F-регистре.

Пример:

```
movlw 0x56      ;Загрузить значение 0x56 в регистр W
movwf 0x20      ;Переместить значение в регистр 0x20
movlw 0xA3      ;Загрузить значение 0xA3 в регистр W
andwf 0x20, 0   ;Логическая операция И над содержимым регистра W и F
```

После выполнения примера значение 0x02 будет находиться в регистре W.

Побитную логическую операцию И (логического умножения) над аргументами W и F при присвоении результата W можно представить на примере следующей таблицы истинности.

W	F	W
0	0	0
0	1	0
1	0	0
1	1	1

Число 1: 0x56 B'01010110'
 Число 2: 0xA3 B'10100011'
 Логическая операция: И
 Результат: 0x02 B'00000010'

Команда: `andlw`

Назначение: Побитная логическая операция И (AND) над содержимым регистра W и 8-битной константой k

Синтаксис: `andlw k`

Изменяемые флаги: Z

По этой команде, как по команде `andwf`, будет выполняться побитная операция И над содержимым регистра W и 8-битной константы k. Эта команда может использоваться, например, чтобы очищать отдельные биты. Называют это также *маскированием*. В следующем примере очищаются 4 старших бита регистра W.

Пример:

```
movlw 0xA7      ;Загрузить значение 0xA7 в регистр W
andlw 0x0F      ;Операция И над содержимым регистра W и константой 0x0F
```

Результат этой операции равен 0x07 и сохраняется в регистре W.

W	k	W
0	0	0
0	1	0
1	0	0
1	1	1

Число 1: 0xA7 B'10100111'
 Константа: 0x0F B'00001111'
 Логическая операция: И
 Результат: 0x07 B'00000111'

Команда: `iorwf`

Назначение: Побитная логическая операция ИЛИ (OR) над содержимым регистра W и регистра, адрес которого указан параметром f, т. е. F-регистра

Синтаксис: `iorwf f,d`

Изменяемые флаги: Z

С помощью команды `iorwf` можно выполнять побитовую логическую операцию ИЛИ над содержимым регистра W и содержимым F-регистра. Результат помещается, в зависимости от параметра d, в регистр W или регистр, указанный f. При d = 0 — результат в регистре W и при d = 1 — в F-регистре.

Пример:

```
movlw 0x5F      ;Загрузить значение 0x5F в регистр W
movwf 0x20      ;Переместить содержимое регистра W по адресу 0x20
movlw 0xAC      ;Загрузить значение 0xAC в регистр W
iorwf 0x20, 1   ;Операция ИЛИ над содержимым регистра W и регистра 0x20
```

После выполнения команд значение 0xFF будет находиться в регистре, адрес которого указан параметром f (F-регистре).

W	F	W
0	0	0
0	1	1
1	0	1
1	1	1

Число 1: 0x5F B'01011111'

Константа: 0xAC B'10101100'

Логическая операция: ИЛИ

Результат: 0xFF B'11111111'

Команда: `iorlw`

Назначение: Побитная логическая операция ИЛИ (OR) над содержимым регистра W и 8-битной константой k

Синтаксис: `iorlw k`

Изменяемые флаги: Z

Как и в предыдущей команде, над двумя значениями будет выполняться побитная операция ИЛИ. Различие состоит в том, что используются содержимые не двух регистров, а одного регистра W и константы. Эта команда очень хорошо может использоваться для установки отдельных битов в регистре. Следующий пример показывает, как оба старших бита устанавливаются в 1, а все остальные биты остаются неприкосновенными.

Пример:

```
movlw 0x25      ;Загрузить значение 0x25 в регистр W
iorlw 0xC0      ;Установить оба старших бита в 1
```

После выполнения команд приведенного примера в регистре W будет храниться значение равное 0xE5.

W	k	W
0	0	0
0	1	1
1	0	1
1	1	1

Число 1: 0x25 B'00100101'

Константа: 0x C0 B'11000000'

Логическая операция: ИЛИ

Результат: 0x E5 B'11100101'

Команда: **xorwf**

Назначение: Побитная логическая операция "исключающее ИЛИ" (XOR) над содержимым регистра W и регистром, адрес которого указан параметром f

Синтаксис: xorwf f,d

Изменяемые флаги: Z

При побитной логической операции "исключающее ИЛИ" (XOR) биты результата будут равны 1, если биты операндов разные, или равны 0, при равенстве битов операндов. В этой команде операндами будут содержимое регистра W и содержимое регистра, адрес которого указан параметром f.

Пример:

```

movlw 0x3A      ;Загрузить значение 0x3A в регистр W
movwf 0x20      ;Переместить это значение в F-регистр 0x20
movlw 0x4B      ;Загрузить значение 0x4B в регистр W
xorwf 0x20, 0    ;Побитная операция XOR над содержимым
                  ;регистра W и регистра с адресом 0x20

```

Результат логической операции равен 0x71 и сохраняется в регистре W, поскольку d = 0. Если бы параметр d был бы равен 1, то результат находился бы в регистре с адресом 0x20.

W	F	W
0	0	0
0	1	1
1	0	1
1	1	0

Число 1: 0x3A B'00111010'

Число 2: 0x4B B'01001011'

Логическая операция: "исключающее ИЛИ"

Результат: 0x 71 B'01110001'

Команда: **xorlw**

Назначение: Побитная логическая операция "исключающее ИЛИ" (XOR) над содержимым регистра W и 8-битной константой k

Синтаксис: **xorlw k**

Изменяемые флаги: **Z**

Для выполнения побитной логической операции "исключающее ИЛИ" над содержимым регистра W и константой существует команда **xorlw**. При этом результат сохраняется в регистре W. С помощью этой команды можно проверить, например, соответствует ли содержимое регистра W заявленному значению. В приведенном примере (для упрощения) значение 0xD4 с помощью команды **movlw** загружается в регистр W. В реальной программе значение могло бы поступить откуда-нибудь, например, из последовательного порта.

Пример:

```
movlw 0xD4      ;Загрузить значение 0xD4 в регистр W
xorlw 0xD6      ;Проверить, равно ли содержимое регистра W значению 0xD6
```

После того как над числами 0xD4 и 0xD6 была выполнена функция "исключающее ИЛИ", результирующее значение 0x02 заносится в регистр W. Это значение не равно 0 и, следовательно, оба значения не идентичны. При равенстве проверяемых значений в регистре состояния установился бы признак нулевого результата.

Побитную логическую операцию "исключающее ИЛИ" над аргументами W и k при присвоении результата W можно представить на примере следующей таблицы истинности.

W	k	W
0	0	0
0	1	1
1	0	1
1	1	0

```
Число 1:      0xD4      B'11010100'
Константа:    0xD6      B'11010110'
Логическая операция:  "исключающее ИЛИ"
Результат:    0x02      B'00000010'
```

Команда: `clrf`

Назначение: Очистить содержимое регистра, адрес которого указан параметром f

Синтаксис: `Clrf f`

Изменяемые флаги: Z

Содержимое регистра, адрес которого указан параметром f, очищается командой `clrf`. При этом совершенно не важно, какое значение было раньше в регистре, после выполнения команды в регистре будут только лишь нули.

Пример:

```
movlw 0x34      ;Загрузить значение 0x34 в регистр W
movwf 0x20      ;Переместить значение в регистр с адресом 0x20
clrf 0x20       ;Очистить содержимое регистра с адресом 0x20
```

После выполнения приведенной последовательности команд в регистр, который имеет адрес 0x20, заносится значение 0x00.

Команда:	clrw
Назначение:	Очистить содержимое регистра W
Синтаксис:	clrw
Изменяемые флаги:	Z

Чтобы очистить содержимое регистра W, или иначе записать в него нули, можно использовать команду `clrw`. В принципе можно было бы очистить содержимое регистра W с помощью команды `movlw 0x00`. Разумеется, в этом случае команда `movlw` не оказывает влияния на флаг нулевого результата Z в регистре состояния. Напротив, после выполнения команды `clrw` флаг Z может измениться, что очень важно, в случае когда заканчивают подпрограмму и хотят проверить, равно ли нулю возвращенное значение в регистре W, или же когда в регистр W записывается значение, отличное от нуля. Иначе пришлось бы применить еще одну дополнительную команду, с помощью которой нужно было бы проверить содержимое регистра W.

Пример:

```
movlw 0x56      ;Загрузить значение 0x56 в регистр W
movlw 0x00      ;Очистить содержимое регистра W командой movlw

movlw 0x56      ;Загрузить значение 0x56 в регистр W
clrw           ;С помощью команды clrw очистить содержимое регистра W
```

После выполнения этих двух примеров содержимое регистра W будет одинаково и равно 0x00, однако только во втором примере будет установлен флаг результата Z.

Команда:	comf
Назначение:	Инвертировать все биты регистра, адрес которого указан параметром f
Синтаксис:	Comf f,d
Изменяемые флаги:	Z

Команда `comf` (от англ. Complement) переводится как "дополнение". После выполнения команды получается значение, которое дополняет заявленное значение до 1, т. е. когда нули исходного числа заменяются единицами и на-

оборот. Поскольку такое наименование несколько "туманно", то проще можно сказать, что значение инвертируется. Если взять какое-либо двоичное число и сложить его с инверсным значением этого числа, то получится двоичное число, которое будет состоять только из одних единиц (0xFF). Для представления как положительных, так и отрицательных чисел отводится дополнительный знаковый разряд, который является старшим разрядом числа. У положительных чисел он равен 0, а у отрицательных — 1. Число ноль имеет в прямом коде два представления: 00...0 и 10...0. Таким образом, чтобы из положительного числа получить отрицательное, нужно к дополнению прибавить лишь 1 ($+4 = 0x04 \rightarrow$ инвертируют $\rightarrow 0xFB \rightarrow$ добавляют 1 $\rightarrow 0xFC = -4$).

Пример:

```
movlw 0x5C      ;Загрузить значение 0x5C в регистр W
movwf 0x20      ;Переместить значение в регистр с адресом 0x20
comf 0x20, 0     ;Инвертировать содержимое регистра с адресом 0x20
```

После выполнения примера в регистре W будет значение равное 0xA3. Если бы дополнительно выполнить команду `addwf 0x20`, то в регистре W будет храниться число 0xFF.

Команда: `bcf`

Назначение: Очистить бит b регистра, адрес которого указан параметром f

Синтаксис: `bcf f,d`

Изменяемые флаги: Нет

Поскольку очень часто требуется, чтобы был сброшен отдельный бит числа, т. е. в этом бите было установлено значение 0, то для этого сделали специальную команду `bcf` (от англ. bit clear). В этой команде указывается регистр и бит, который должен быть сброшен. Кроме того, соответствующая позиция бита показывается в регистре. В программе с помощью директивы `equ` биту можно назначить имя, посредством которого к нему можно будет потом обращаться.

Пример:

```
BIT4 EQU D'4'    ;Присвоить имя 'BIT4' числу 4
movlw 0x3F      ;Загрузить значение 0x3F в регистр W
movwf 0x20      ;Переместить это значение в регистр с адресом 0x20
bcf 0x20, BIT4   ;Очистить бит 4 в регистре с адресом 0x20
```

Как ожидалось, бит 4 очищается и значение 0x2F заносится в регистр с адресом 0x20.

Команда:	bsf
Назначение:	Установить бит b регистра, адрес которого указан параметром f
Синтаксис:	bsf f,d
Изменяемые флаги:	Нет

Для установки определенного бита в регистре можно использовать команду **bsf** (от англ. bit set). Эта команда устанавливает указанный бит в соответствующем регистре. Следует учитывать, что в цифровой технике нумерация разрядов начинается с 0, при этом обыкновенно нулевым битом является бит, стоящий крайним справа и называется *младшим разрядом двоичного числа* (LSB — Least Significant Bit).

Пример:

```
movlw 0x2F      ;Загрузить значение 0x2F в регистр W
movwf 0x20      ;Переместить это значение в регистр с адресом 0x20
bsf 0x20, 4     ;Установить в единицу 4 бит в регистре с адресом 0x20
```

После выполнения последовательности команд в регистре с адресом 0x20 будет храниться значение равное 0x3F.

2.2.4. Команды сдвига

Команда:	rlf
Назначение:	Выполнить циклический сдвиг влево на один бит содержимого регистра, адрес которого указан параметром f (F-регистра). Сдвиг осуществляется с использованием бита переноса C регистра состояния (STATUS)
Синтаксис:	rlf f,d
Изменяемые флаги:	C

Бит C (от англ. Carry) в регистре состояния (STATUS) — это бит переноса. В нем, в принципе, может сохраняться 9-й бит значения. Бит C используется, если 8 битов регистра недостаточно. После выполнения команды **rlf** (от англ. Rotate Left) биты циклически сдвигаются влево на один разряд (рис. 2.1). При этом *старший разряд двоичного числа* (MSB — Most Significant Bit), т. е. бит 7, выдвигается в бит переноса C, который находится в регистре состояния STATUS. Прежнее значение бита C выдвигается и сохраняется в F-регистре на месте младшего бита 0 (LSB). Таким образом, осуществляется циклический сдвиг содержимого регистра влево на один разряд. В целом такой сдвиг соответствует умножению содержимого регистра на 2.

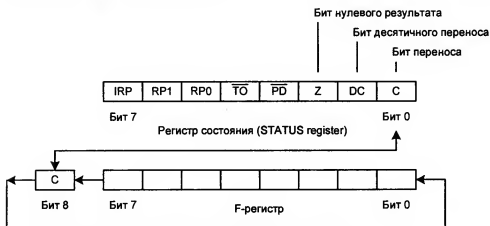


Рис. 2.1. Циклический сдвиг бита влево

Пример:

```
movlw 0x8C      ;Загрузить значение 0x8C в регистр W
movwf 0x20      ;Переместить это значение в регистр с адресом 0x20
rlf 0x20, 0     ;Умножить содержимое регистра с адресом 0x20 на 2
```

При умножении числа 0x8C (D'140') на 2, или иначе сдвиге на один разряд влево без учета бита переноса C, в регистре W сохранялся бы результат равный 0x18 (D'24'). Но поскольку фактически в итоге результат должен быть больше чем 255, нужно обязательно в качестве старшего бита учитывать бит переноса C (Carry) из регистра состояния (STATUS). Если же учесть бит переноса вместе со значением регистра W, то в итоге получают действительный полный результат операции умножения: $0x8C \times 2 = 0x118 = D'280'$ ($140_{10} \times 2 = 280_{10}$).

Если требуется умножить это значение еще раз на 2, чтобы получить умножение на 4, нужно временно сохранить бит переноса C в другом регистре и очистить его после этого в регистре состояния. После циклического сдвига на один разряд влево бит переноса (т. е. 0) записывается в младший разряд двоичного числа (результата) в регистре с адресом, указанным f. Поскольку получившееся число превышает значение 255, то при прочтении результата надо учитывать текущий бит переноса и бит переноса, сохраненный в другом регистре: $0x8C \times 4 = 0x230 = D'560'$ ($140_{10} \times 4 = 560_{10}$).

С помощью команды `rlf` достаточно просто могут считываться последовательные данные, поступающие на соответствующий вывод порта. Для этого запрашивается вывод порта, и его значение перемещается в младшую позицию регистра. После циклического сдвига регистра можно снова считывать

значение с вывода порта и записывать его на место нулевого бита регистра. Таким образом, при 8-кратном повторе такой процедуры в регистре будет получено 8-битное значение последовательных данных.

Команда: `rrf`

Назначение: Выполнить циклический сдвиг вправо на один бит содержимого регистра, адрес которого указан параметром `f` (F-регистра). Сдвиг осуществляется с использованием бита переноса `C` регистра состояния (STATUS)

Синтаксис: `rrf f,d`

Изменяемые флаги: `C`

Команду `rrf` (от англ. Rotate Right) используют для деления на 2. При этом биты циклически сдвигаются или иначе перемещаются на одну позицию вправо (рис. 2.2). Если требуется разделить на 2, то нужно обращать внимание на то, чтобы бит `C` был бы установлен перед делением, поскольку он перемещается на позицию старшего бита (MSB) F-регистра. Младший бит (LSB) этого же регистра выдвигается и помещается в позицию бита переноса `C` регистра состояния.

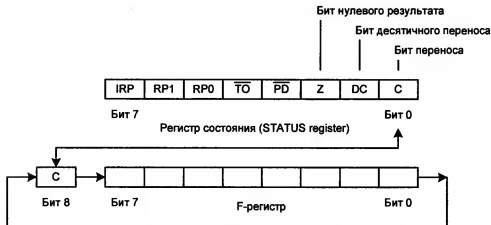


Рис. 2.2. Циклический сдвиг вправо

Пример:

```
movlw 0x73      ;Загрузить значение 0x73 в регистр W
movwf 0x20      ;Переместить это значение в F-регистр с адресом 0x20
rrf 0x20, 0     ;Сдвинуть все биты F-регистра на одну позицию вправо
```

Если перед выполнением операции сдвига бит переноса был установлен, то в регистре W будет значение равное $0xB9$ ($0x173 : 2 = 0xB9$ или $371_{10} : 2 = 185_{10}$). Если же бит переноса будет сброшен, то в регистре W появится неправильный результат — $0x39$ (57_{10}).

Команду циклического сдвига вправо `rrf` очень хорошо использовать для последовательного вывода данных. Значение разрядностью 8 бит загружается в регистр, затем выводят младший бит в порт, потом содержимое регистра сдвигают на одну позицию вправо и снова выводят младший бит. Процедуру повторяют до тех пор, пока все биты не будут выведены.

Команда: `movlw`
Назначение: Переслать константу k в регистр W
Синтаксис: `movlw k`
Изменяемые флаги: Нет

Чтобы загрузить константу в регистр W, используют команду `movlw` (от англ. Move). С помощью этой команды в регистр W можно загружать любое значение от 0 до 255 и затем продолжить с ним работу.

Пример:

```
movlw D'129'      ;Загрузить десятичное значение 129 в регистр W
```

Это очень простая и часто используемая команда, которая применяется в каждой программе на ассемблере минимум один раз, поскольку для последующей обработки того или иного значения всегда вначале нужно его загрузить в регистр W.

Команда: `movwf`
Назначение: Выполнить пересылку содержимого регистра W в регистр, адрес которого указан параметром f (F-регистр)
Синтаксис: `movwf f`
Изменяемые флаги: Нет

Если требуется переместить некоторое значение из регистра W в F-регистр, чтобы вывести его, например, в порт вывода, то нужно применить команду `movwf`, которая выполняет пересылку содержимого регистра W в регистр, указанный f.

Пример:

```
movlw 0xF2        ;Загрузить в регистр W значение 0xF2  
movwf 0x20        ;Сохранить его в регистре с адресом 0x20
```

Эта команда используется так же часто, как и команда `movlw`, поскольку в любой программе все значения постоянно сохраняются в RAM-памяти, чтобы затем можно было продолжить с ними работу или применить к ним ту или иную функцию. С помощью этой команды, например, осуществляют пересылку значения в регистр `TRISA`, который предназначен для управления направлением передачи данных порта ввода/вывода.

Команда: `movf`

Назначение: Выполнить пересылку содержимого регистра, адрес которого указан параметром `f` (F-регистра), в регистр адресата

Синтаксис: `movf f, d`

Изменяемые флаги: `Z`

Если для дальнейшей обработки требуется значение, сохраненное ранее в RAM-памяти, получают его назад пересылкой в регистр `W` с помощью команды `movf`. Однако у этой команды есть маленькая особенность: в команде можно указать, где значение должно сохраняться. При `d = 0` значение, при желании, пересылается в регистр `W`, а при `d = 1` он сохраняется в регистре с адресом, указанным `f` (F-регистре). Поскольку оно и так уже находится в F-регистре, то это, на первый взгляд, вроде бы бессмысленно. Однако это хорошая возможность того, чтобы проверить, содержит ли указанный F-регистр нулевое значение, т. к. после вызова этой команды в регистре состояния может измениться бит (флаг) нулевого результата `Z`.

Пример:

```
movlw 0x24      ;Загрузить значение 0x24 в регистр W
movwf 0x20      ;Сохранить это значение в F-регистр с адресом 0x20
movf 0x20, 1     ;Проверить, будет ли содержимое регистра 0x20 равно 0
btfss STATUS, Z ;Если значение равно 0, то пропустить следующую команду
movf 0x20, 0     ;Если значение не равно 0, то переслать его в регистр W
```

С помощью команды `movf` можно, например, проверить, досчитал ли счетчик, считающий в обратном порядке, вплоть до нуля.

Команда: `swapf`

Назначение: Поменять местами старший и младший полубайты регистра, адрес которого указан параметром `f`

Синтаксис: `swapf f, d`

Изменяемые флаги: Нет

Если необходимо поменять местами старшие 4 бита 8-битного слова с младшими 4 битами, то это можно выполнить, воспользовавшись командой `swarf` (от англ. *Swar*). В противоположность другим командам сдвига эта команда используется достаточно редко. Возможное применение этой команды связано со считыванием 4 битов порта ввода. Если на выводы RB4—RB7 поступает некоторое значение, то порт может читаться, а полубайты меняться и интерпретироваться после маскирования. Никакого 4-кратного циклического сдвига тогда не потребуется, а можно воспользоваться только этой одной командой.

Пример:

```
swarf PORTB, 0      ;Читать порт В и обменивать полубайты
andlw 0x0F          ;Маскировать старшие 4 бита
```

После выполнения обеих команд шестнадцатеричное значение старших 4 битов порта В будет в регистре W. Если значение B'01101100' находится в регистре PORTB, результат обеих команд — число 0x06 в регистре W.

2.2.5. Арифметические команды

Команда: `addwf`

Назначение: Сложить содержимое регистра W и регистра, адрес которого указан параметром f

Синтаксис: `addwf f, d`

Изменяемые флаги: C, DC, Z

Команда `addwf` используется для сложения 2 значений. После выполнения команды результат будет находиться в зависимости от параметра d в регистре W (d = 0) или в регистре, адрес которого указан параметром f (F-регистре) (d = 1).

Пример:

```
movlw 0x39          ;Загрузить значение 0x39 в регистр W
movwf 0x20          ;Переместить это значение в регистр с адресом 0x20
movlw 0x14          ;Загрузить значение 0x14 в регистр W
addwf 0x20, 1       ;Сложить содержимое регистров W и F-регистром
```

После сложения 0x39 и 0x14 результат 0x4D будет сохранен в регистре, адрес которого 0x20 указан параметром f, т. е. иначе F-регистре. В этом примере сложение выполняется без переноса в старших разрядах. Если же при сложении двух значений в результате получается число, которое больше 255, то в

регистре состояния STATUS будет установлен флаг переноса C. В том случае, когда при сложении получается число 0x100 (D'256'), то еще дополнительно устанавливается флаг нулевого результата Z.

Команда:	addlw
Назначение:	Сложить содержимое регистра W с 8-разрядной константой k
Синтаксис:	addlw k
Изменяемые флаги:	C, DC, Z

Если константа прибавляется лишь к значению, которое находится в настоящее время в регистре W, то это может быть выполнено с помощью команды addlw.

Пример:

```
movlw 0x39      ;Загрузить значение 0x39 в регистр W
addlw 0x0A      ;Добавить константу 0x0A к этому значению
```

После сложения обоих чисел в регистре W будет значение 0x43. В этом примере выполняется перенос в 4-й бит и поэтому в регистре состояния устанавливается флаг десятичного переноса DC (Digit Carry).

Команда:	subwf
Назначение:	Вычесть содержимое регистра W из содержимого регистра, адрес которого указан параметром f
Синтаксис:	subwf f, d
Изменяемые флаги:	C, DC, Z

Команда subwf выполняет вычитание содержимого регистра W из указанного f регистра (F-регистра). Результат влияет на указанные флаги в регистре состояния и сохраняется в зависимости от значения параметра d в регистре W (d = 0) или в регистре, адрес которого указан параметром f (d = 1).

Пример:

```
movlw 0x71      ;Загрузить значение 0x71 в регистр W
movwf 0x20      ;Переместить его в регистр с адресом 0x20, указанным f
movlw 0xA7      ;Загрузить значение 0xA7 в регистр W
subwf 0x20, 1    ;Вычесть содержимое регистра W из содержимого
                  ;F-регистра, указанного параметром f (F - W -> F)
```


Результат вычитания $0xCA$ ($D'202'$) будет сохранен в F-регистре.

$0x71$ ($D'113'$) — $0xA7$ ($D'167'$) = $0xCA$ ($D'-54'$) → Флаг переноса равен 0

$0x71$ ($D'113'$) — $0x37$ ($D'55'$) = $0x3A$ ($D'58'$) → Флаг переноса равен 1

При вычитании надо обратить внимание, что в этом случае используются биты заема, а не биты переноса. Поэтому число, при котором разряд переноса установлен на 0, интерпретируется как отрицательное число, а результат находится как двоичное дополнение в регистре назначения. Для получения отрицательного числового значения нужно результат проинвертировать и добавить 1.

```
comf 0x20, 1      ;Инвертировать содержимое F-регистра с адресом 0x20
incf 0x20, 1      ;Прибавить 1 к содержимому F-регистра
```

После формирования точного дополнения в двоичной системе счисления получают величину отрицательного числа.

$0xCA$ ($D'202'$) → Образовать дополнение в двоичной системе счисления → $0x36$ ($D'54'$)

Команда: `sublw`

Назначение: Вычесть содержимое регистра W из 8-разрядной константы k

Синтаксис: `sublw k`

Изменяемые флаги: C, DC, Z

У команды `sublw` имеется маленькая ловушка, на которую нужно обращать внимание. Здесь содержимое регистра W вычитается из константы и не наоборот.

Пример:

```
movlw 0x43      ;Загрузить регистр W значением 0x43
sublw 0x83      ;Вычесть содержимое регистра W из константы k (k - W -> W)
```

Результатом этого вычитания будет значение $0x40$, которое будет находиться в регистре W.

Команда: `incf`

Назначение: Прибавить 1 к содержимому регистра, адрес которого указан параметром f

Синтаксис: `incf f, d`

Изменяемые флаги: Z

Команда `incf` — это часто используемая команда. При ее вызове содержимое указанного `f` регистра увеличивается на 1 (инкрементируется).

Пример:

```
movlw 0xFE      ;Загрузить значение 0xFE в регистр W
movwf 0x20      ;Переместить это значение в регистр с адресом 0x20
incf 0x20, 1     ;Прибавить к содержимому этого регистра 1
incf 0x20, 1     ;Увеличить это содержимое еще раз на 1
```

После выполнения первой команды `incf` в регистре, адрес `0x20` которого указан параметром `f`, т. е. F-регистре, будет храниться значение `0xFF`. Это самое большое отображаемое 8-битное значение. После следующей команды `incf` этот регистр будет "переполнен" и в нем будут только лишь нули (`0x00`). Поэтому после выполнения этой команды устанавливается флаг нулевого результата `Z`. Разумеется, команда не оказывает влияния на флаг десятичного переноса (`DC`) и флаг переноса (`C`), поскольку речь не идет о переносе как при сложении.

Команда: `decf`

Назначение: Вычесть 1 из содержимого регистра, адрес которого указан параметром `f`

Синтаксис: `decf f, d`

Изменяемые флаги: `Z`

Для уменьшения содержимого регистра на 1 применяют команду `decf`. Эта команда часто используется, чтобы организовать счетчик цикла или таймер. Для этого значение, равное количеству циклов, загружается в регистр и при каждом прохождении цикла уменьшается на 1. При проверке флага нулевого результата можно определить окончание счета.

Пример:

```
movlw 0x02      ;Загрузить значение 0x02 в регистр W
movwf 0x20      ;Переместить содержимое в регистр с адресом 0x20
decf 0x20, 1     ;Уменьшить на 1 содержимое этого регистра
decf 0x20, 1     ;Уменьшить содержимое еще раз на 1
```

После выполнения последовательности команд значение `0x00` будет находиться в регистре с адресом `0x20`, а в регистре состояния `STATUS` будет установлен флаг нулевого результата `Z`. Также как и в случае с командой `incf`, флаги `DC` и `C` не изменяются.

2.2.6. Команды передачи управления

Команда:	<code>goto</code>
Назначение:	Выполнить безусловный переход, т. е. переход на указанный адрес
Синтаксис:	<code>goto k</code>
Изменяемые флаги:	Нет

При достижении в программе команды `goto`, выполняющей *безусловный переход* (англ. unconditional branch), осуществляется передача управления на указанный адрес (метку) без проверки какого-либо условия, т. е. безусловно. Условие же, например, могло бы выглядеть так: "Перейти на выполнение команды, если определенный бит равен 0". Поэтому эта команда перехода запускается сразу без проверки ранее определенного состояния. Указанная константа в случае PIC16F876A может иметь значение от 0 и до 2047_{10} ($0x0000—0x07FF$). Это значение представляет собой младшие 11 битов адреса следующей команды. Два других бита, которые требуются для получения 13-битного адреса, поступают из двух старших разрядов специального 5-разрядного регистра PCLATH (рис. 2.3). Поэтому при обращении ко всей области памяти, состоящей из 4 страниц пронумерованных с 0 по 3, или при изменении адреса через значения 2048_{10} ($0x0800$ начальный адрес страницы 1), 4096_{10} ($0x1000$ начало страницы 2) и 6144_{10} ($0x1800$ начало страницы 3) необходимо обеспечить, чтобы в регистре PCLATH старшие его разряды 3 и 4 имели соответствующие значения. Поскольку содержимое программного счетчика PC (англ. Program Counter) получается из 2 частей, то для выполнения этой команды требуются 2 командных цикла.

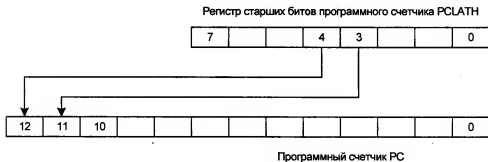


Рис. 2.3. Команда перехода GOTO

Пример:

```

main
    movlw 0x55      ;Загрузить в регистр W значение 0x55
    movwf PORTC     ;Вывести данные в порт C
    movlw 0xAA      ;Загрузить значение 0xAA в регистр W
    movwf PORTC     ;Вывести данные в порт C
    goto main       ;Перейти к метке main (бесконечный цикл)

```

В данном примере на вывод RC0 (0 бит порта ввода/вывода C) микросхемы PIC16F876A попеременно поступает то низкий, то высокий логический уровень. Программа работает в бесконечном цикле. В команде `goto` адрес перехода указывается почти во всех случаях не в форме числа, а в форме метки перехода (label). Это чрезвычайно облегчает программирование, поскольку иначе необходимо было бы точно знать и учитывать все адреса программы для верного указания адреса перехода в виде числа в команде `goto`.

Команда: `call`

Назначение: Перейти на выполнение подпрограммы

Синтаксис: `call k`

Изменяемые флаги: Нет

Команда `call` очень похожа на команду `goto`, т. к. она также является командой безусловного перехода. Особенность команды `call` состоит в том, что перед самым переходом гарантированно сохраняется текущее содержимое программного счетчика, что в дальнейшем позволяет возвращаться в исходную точку программы. Поэтому команда `call` используется для вызова подпрограммы. В ходе выполнения команды содержимое программного счетчика сохраняется в стеке, а с помощью команды `return`, которая является последней в подпрограмме, снова загружается в программный счетчик. Одиннадцать бит адреса перехода загружаются из кода команды в 13-разрядный счетчик команд ($k \rightarrow PC<10:0>$), а два старших бита получают из регистра PCLATH ($PCLATH<4:3> \rightarrow PC<12:11>$). При выполнении этой команды требуется два командных цикла.

Пример:

```

mask      ;Метка начала подпрограммы
    andlw 0x03 ;Маскирование обоих младших битов содержимого регистра W
    return    ;Возврат из подпрограммы
main      ;Метка начала основной программы
    movlw 0x55 ;Загрузить значение 0x55 в регистр W
    call mask  ;Вызвать подпрограмму
    movwf PORTC ;Вывести значения в порт C
    goto main  ;Перейти на метку main (бесконечный цикл)

```

При выполнении подпрограммы с меткой `mask` все биты вплоть до двух младших сбрасываются в 0. Оба младших бита сохраняют свое прежнее значение, т. е. они маскируются. После выполнения маскирования осуществляется возврат в основную программу на команду, следующую за командой `call`, которая вызывала подпрограмму.

Команда:	return
Назначение:	Выполнить возврат из подпрограммы
Синтаксис:	<code>Return</code>
Изменяемые флаги:	Нет

Команда `return` предоставляет возможность для возврата из подпрограммы. После вызова команды берется значение, которое хранится на вершине стека, и затем загружается в программный счетчик. Далее программа продолжает свою работу начиная с команды, адрес которой хранился в стеке, т. е. команды следующей за командой `call`, вызвавшей подпрограмму. Поскольку стековая память может сохранять только максимум 8 адресов возврата, количество вложенных вызовов подпрограммы соответственно ограничено.

Пример:

```
rotate      ;Метка начала подпрограммы rotate
    rrf 0x20, 0 ;Циклически сдвинуть вправо биты содержимого
                ;регистра 0x20, а результат поместить в регистр W
    return   ;Возврат из подпрограммы rotate
mask        ;Метка начала подпрограммы mask
    andwf 0x20, 1 ;Маскировать содержимое регистра W значением из
                ;F-регистра с адресом 0x20, поместив в него результат
    call rotate ;Вызов подпрограммы rotate
    return     ;Возврат из подпрограммы mask
main        ;Главная программа
    movlw 0xAA ;Загрузить значение 0xAA в регистр W
    movwf 0x20 ;Переместить это значение в F-регистр с адресом 0x20
    movlw 0x0E ;Загрузить в регистр W значение 0x0E
    call mask  ;Вызвать подпрограмму mask
    movwf PORTC ;Вывести значения регистра W в порт C
    goto main
```

Приведенный пример демонстрирует вложенный вызов подпрограмм. Используются 2 уровня стека для сохранения адресов возврата из подпрограмм. После выполнения всей программы в регистре W будет храниться значение равное 0x05. Это результат побитной логической операции И числа 0xAA

с числом 0x0E и дальнейшего циклического сдвига битов полученного результата вправо.

Команда:	retlw
Назначение:	Выполнить возврат из подпрограммы с загрузкой константы k в регистр W
Синтаксис:	retlw k
Изменяемые флаги:	Нет

Команда **retlw** предлагает еще одну возможность для выполнения возврата из подпрограммы. В данном случае помимо самого возврата константа k, указанная в команде, сохраняется в регистре W. Это же действие возможно, если сначала с помощью одной команды выполнить сохранение значения в регистре W, а затем, воспользовавшись командой **return**, возвратиться из подпрограммы. Однако, применяя команду **retlw**, можно сэкономить один командный цикл, что может быть очень важно в некоторых программах. Кроме того, эту команду удобно использовать, чтобы программе, вызывающей подпрограмму, предоставлять значение, которое сообщало бы о том, было ли успешным выполнение подпрограммы или произошла ошибка. Так, в следующем примере подпрограмма при безошибочном выполнении возвращает 0, а в случае ошибки — 1.

Пример:

```

rotate                ;Метка начала подпрограммы rotate
    rlf 0x20, 1        ;Циклический сдвиг влево содержимого регистра 0x20
    btfsc STATUS, C    ;Проверить бит переноса C в регистре состояния
    retlw 1            ;Поместить 1 в рег. W и вернуться из подпрограммы,
                        ;если C=1, т. е. имеется переполнение, ошибка
    retlw 0            ;Поместить 0 в рег. W и вернуться из подпрограммы,
                        ;если C=0, т. е. нет переполнения
main                  ;Главная программа
    movlw 0x55         ;Загрузить начальное значение 0x55 в регистр W
    movwf 0x20         ;Переместить значение в регистр 0x20
loop                  ;Метка начала цикла
    call rotate        ;Вызов подпрограммы rotate
    andlw 0x01         ;Выполнить операцию И с содержимым W и числом 0x01
    btfss STATUS, Z    ;Проверить, установлен ли флаг нулевого результата
    goto main         ;Ошибка, Z=1, определено переполнение
    goto loop         ;OK, Z=0, Повторить цикл

```

Основная программа проверяет значение в регистре W после возврата из подпрограммы и в случае отсутствия ошибок цикл повторяется снова. Если

же в подпрограмме встретилось переполнение ($C = 1$), то программа начинается с самого начала с метки `main`.

Команда:	<code>retfie</code>
Назначение:	Выполнить возврат из подпрограммы обработки прерывания и разрешить прерывания
Синтаксис:	<code>retfie</code>
Изменяемые флаги:	Нет

Прерывание (англ. Interrupt) может произойти в любой момент и соответственно в любой точке программы при ее выполнении. Следовательно, значение программного счетчика при этом может иметь заранее неизвестное значение. Поэтому если случается прерывание, то текущее содержимое программного счетчика должно быть сохранено в стеке, и только после этого происходит переход на программу обработки прерывания. После выполнения программы обработки прерывания нужно снова перейти на ту команду программы, которая выполнялась бы не будь прерывания. Во время выполнения программы обработки прерывания никакие новые прерывания не допускаются. Поэтому после завершения программы обработки прерывания должен отмениться запрет на прерывания. Это и выполняется с помощью команды `retfie`. Команда перемещает значение из вершины стека в программный счетчик и устанавливает в регистре `INTCON` бит `GIE` (Global Interrupt Enable), разрешая тем самым все прерывания.

Пример:

```

w_temp EQU 0x70          ;Определить регистры для временного сохранения
status_temp EQU 0x71     ;содержимого регистра W и регистра состояния
ORG 0x004                ;Адрес вектора прерывания
movwf w_temp             ;Сохранить содержимое регистра W
movf STATUS, W           ;Переместить сод. регистра состояния в регистр W
movwf status_temp        ;Сохранить содержимое регистра STATUS
                           ;Ниже показан код для программы обработки прерывания
movf PORTB, W            ;Читать порт B
movwf PORTC              ;Передать значение порта B далее в порт C
movlw b'00001000'        ;Возвратить биты прерывания
movwf INTCON
movf status_temp, W      ;Получить копию регистра состояния STATUS
movwf STATUS             ;Восстановить значение регистра состояния
swapf w_temp, F          ;Сохранить данные назад в регистр W
swapf w_temp, W
retfie                   ;Возврат из прерывания

```

```

main                                ;Главная программа
    movlw b'10001000'               ;Разрешить прерывание, которое вызывается
    movwf INTCON                    ;после изменения сигнала на входах RB7:RB4 PortB
loop
    movlw 0x05                      ;Загрузить значение 0x05 в регистр W
    movwf PORTC                     ;Вывести значения в порт C
    movlw 0x0A                      ;Загрузить значение 0x0A в регистр W
    movwf PORTC                     ;Вывести значения в порт C
    goto loop                       ;Повторить цикл

```

В приведенном примере цикл (loop) в основной программе выполняется многократно. При этом попеременно выводятся значения 0x05 и 0x0A в порт C. Если же сигналы на выходах RB4, RB5, RB6 или RB7 изменяются, то они вызывают прерывание, и программа переходит на программу обработчика прерывания. После возникновения прерывания прежде всего нужно временно сохранить содержимое регистра W и регистра состояния, чтобы после выполнения программы обработки можно было бы снова вернуть их первоначальные значения. В программе обработки прерывания порт В читается и копируется в порт C, затем продолжается основная программа. Само собой разумеется, что также при переходе в программу обработки прерывания в стеке сохраняется значение программного счетчика и восстанавливается обратно из стека в программный счетчик с помощью команды *retfie*. Обменять местами полубайты данных в регистре W с помощью команды *swaph* необходимо, т. к. команда *movf w_temp* снова могла бы повлиять в регистре состояния на флаг нулевого результата. Поскольку при появлении прерывания не известно, какой банк в данный момент активен, временные регистры *w_temp* и *status_temp* должны иметь адрес, который имеется в наличии в каждом банке. Поэтому для регистра *w_temp* выбирается адрес 0x70, а для *status_temp* — 0x71. Следует заметить, что, чтобы гарантировать лучшую наглядность, в приведенном примере процедура инициализации порта опущена.

Команда: *incfsz*

Назначение: Прибавить 1 к содержимому регистра, адрес которого указан параметром *f* (F-регистра), и пропустить следующую команду, если результат равен 0

Синтаксис: *incfsz f, d*

Изменяемые флаги: Нет

Команда *incfsz* (рис. 2.4) принадлежит к условным командам перехода, поскольку переход осуществляется только, если после инкрементирования

результатирующее значение было равно 0. При обработке команды происходит следующее. Содержимое регистра, адрес которого указан параметром *f* (F-регистра), увеличивается на 1, затем проверяется, получился ли результат равный 0 ($0xFF + 1 = 0x00$). Если это действительно так, то следующая команда пропускается или она меняется на команду *nop* (no operation — нет операции). Поэтому в команде используется 2 командных цикла в случае передачи управления. Если же результат не равен 0, то никакого перехода не происходит, а выполняется следующая ближайшая команда, и в этом случае для команды достаточно только одного командного цикла.

После выполнения команды результат записывается в зависимости от параметра *d* в регистр *W* (*d* = 0) или F-регистр (*d* = 1).



Рис. 2.4. Команда перехода *INCFSZ*

Пример:

movlw 0xF9	;Загрузить значение 0xF9 в регистр W
movwf 0x20	;Скопировать значение в регистр с адресом 0x20
counter	;Метка начала цикла
incfsz 0x20, 1	;Увеличить на 1 содержимое регистра с адресом 0x20
goto counter	;Если результат не равен 0, перейти на метку counter

Для демонстрации команды `incfsz` в данном примере к содержимому регистра с адресом 0x20 в цикле прибавляется 1, начиная от значения 0xF9 и заканчивая 0x00 (0xFF + 1). При этом такой цикл, например, может применяться для временной задержки. В описанном примере для обработки команд `incfsz` и `goto` требуются всего 20 командных циклов. Это соответствовало бы при тактовой частоте микроконтроллера 4 МГц задержке равной 20 мкс (поскольку цикл выполнения одной команды состоит из 4 периодов тактовой частоты).

Команда: `decfsz`

Назначение: Вычесть 1 из содержимого регистра, адрес которого указан параметром `f` (F-регистра), и пропустить следующую команду, если результат равен 0

Синтаксис: `decfsz f, d`

Изменяемые флаги: Нет

В отличие от команды `incfsz` при выполнении команды `decfsz` (рис. 2.5) содержимое указанного F-регистра не увеличивается, а уменьшается на 1. Результат также сохраняется в регистре, указанном параметром `d`, т. е. в регистре W при `d = 0` или в F-регистре при `d = 1`. После операции уменьшения на единицу проверяется, равен ли 0 результат. Если это действительно так, то следующая команда пропускается или иначе она меняется на команду `nop`. Поэтому в команде используется 2 командных цикла в случае передачи управления. Если же результат не равен 0, то никакого перехода не происходит, а выполняется следующая ближайшая команда, и в этом случае для команды достаточно только одного командного цикла.

Пример:

movlw 0x09	;Загрузить число 0x09 в регистр W
movwf 0x20	;Загрузить его в регистр 0x20, используя как счетчик
movlw 0xA6	;Загрузить число 0xA6 в регистр W
movwf 0x21	;Загрузить его в регистр 0x21, используя как рабочий
shift	;Метка начала цикла
rlf 0x21, 1	;Циклический сдвиг влево содержимого рабочего рег.

decfsz 0x20, 1 ;Уменьшить значение счетчика на 1
goto shift ;Возвращаться на метку shift до тех пор, пока счетчик ≠ 0



Рис. 2.5. Команда перехода DECFSZ

Содержимое рабочего регистра с адресом 0x21 9 раз побитно циклически сдвигается влево с использованием бита переноса регистра состояния STATUS. После выполнения команд программы в рабочем регистре с адресом 0x21 будет то же самое значение, что и до выполнения сдвигов, т. е. 0xA6.

Команда:

btfs

Назначение:

Проверить бит b в регистре, адрес которого указан параметром f. Пропустить следующую инструкцию, если бит b равен 1

Синтаксис: `btfss f, b`

Изменяемые флаги: Нет

При программировании на ассемблере часто требуется проверить состояние отдельного бита в регистре (рис. 2.6). Например, нужно знать, встретился ли при сложении двух значений перенос. Чтобы проверить это, в регистре состояния можно проверить флаг переноса. В принципе, все регистры состоят из отдельных битов, которые, в зависимости от состояния операции, устанавливаются или сбрасываются.

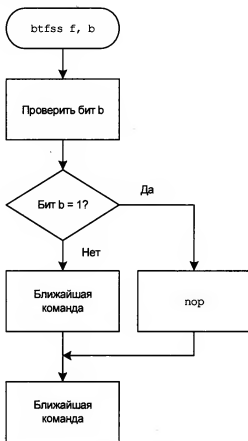


Рис. 2.6. Команда перехода BTFSS

Пример:

<code>movlw 0xD6</code>	<code>;Загрузить значение 0xD6 в регистр W</code>
<code>movwf 0x20</code>	<code>;Скопировать его в регистр с адресом 0x20</code>

```

weiter           ;Метка начала цикла
    movlw d'5'   ;Загрузить в регистр W число d'5'
    addwf 0x20, 1 ;Сложить содержимое регистра W и регистра 0x20
    btfss STATUS, C ;Проверить, был ли перенос
    goto weiter ;если переноса не было, то перейти на метку weiter

```

В примере десятичное значение 5 прибавляется к значению в регистре 0x20 до тех пор, пока не будет переноса. Если будет перенос (бит регистра состояния STATUS C = 1), то цикл оканчивается, и команда goto заменяется на команду пор (нет операции) или иначе просто пропускается.

Команда: `btfsc`

Назначение: Проверить бит b в регистре, адрес которого указан параметром f. Пропустить следующую инструкцию, если бит b равен 0

Синтаксис: `btfsc f, b`

Изменяемые флаги: Нет

Чтобы проверить, 0 или 1 содержит бит в регистре, используют команду `btfsc` (рис. 2.7). При этом если проверяемый бит равен 0, то команда, которая непосредственно стоит за командой `btfsc`, пропускается. Нумерация битов в регистре начинается с 0. Следовательно, старший бит (MSB) — это бит 7, а младший (LSB) — бит 0.

Пример:

```

    movlw 0x00           ;Очистить регистр W
    movwf 0x20           ;Очистить регистр с адресом 0x20
loop           ;Метка начала цикла
    btfsc 0x20, 7        ;Проверить бит 7 в регистре с адресом 0x20
    goto add10           ;Если бит 7 = 1, то перейти на метку add10
add5           ;Если бит 7 = 0, то добавить 5
    addlw d'5'           ;Установить бит 7 в регистре 0x20
    bsf 0x20, 7          ;Перейти на метку ende_add
    goto ende_add
add10          ;Если бит 7 = 1, то добавить 10
    addlw d'10'          ;Очистить бит 7 в регистре 0x20
    bcf 0x20, 7
ende_add
    goto loop           ;Перейти к метке loop

```

В примере программы десятичное значение 5 или 10 попеременно добавляется к содержимому регистра W. Если бит 7 установлен, то к значению в реги-

стре W прибавляется 10, а если бит 7 не установлен (равен 0), то прибавляется только 5.

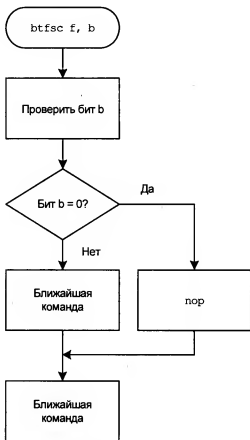


Рис. 2.7. Команда перехода BTFSK

2.2.7. Прочие команды

Команда: `por`

Назначение: Нет операции (пустая операция)

Синтаксис: `por`

Изменяемые флаги: нет

Если команда `por` указывается в программе, то процессор при ее выполнении ничего не делает. Во время командного цикла никакие операции не запуска-

ются. Эта на первый взгляд бессмысленная команда охотно используется, чтобы задавать небольшое время ожидания в программе. Если осуществляется, например, коммуникация с другим модулем, который нуждается в двух командных циклах для ответа на событие, то можно вводить в программу команду `nop` 2 раза и предоставлять, таким образом, второму модулю небольшое время для ответа.

Пример:

```
movlw d'100'      ;Загрузить в регистр W значение 100
movwf 0x20         ;Переместить его в регистр с адресом 0x20
wait2us
nop                ;Ждать 1 командный цикл (при частоте 4 МГц это 1 мкс)
nop                ;Ожидание еще 1 мкс
incf 0x20, 1       ;Увеличить на 1 значение регистра с адресом 0x20
```

В этом примере процессор выполняет операцию инкрементирования (увеличения на 1) только после времени ожидания, равного двум командным циклам.

Команда: `clrwdt`

Назначение: Очистить сторожевой таймер WDT и предделитель, если он подключен к таймеру WDT

Синтаксис: `clrwdt`

Изменяемые флаги: TO, PD

Watchdog переводится дословно "сторожевая собака" процессора. Эта команда — защитная функция микроконтроллера и используется для проверки правильной работоспособности процессора. Сторожевой таймер WDT (Watchdogtimer) устанавливается и работает в режиме обратного отсчета времени в фоновом режиме. Если этот отсчет времени доходит до конца, т. е. происходит переполнение сторожевого таймера WDT, то выполняется сброс, и процессор переводится в начальное состояние. Чтобы процессор не сбрасывался, сторожевой таймер должен периодически очищаться с помощью команды `clrwdt` или устанавливаться снова на значение для запуска. Это очень удобно для вывода процессора из непреднамеренного бесконечного цикла. Процессор из этого цикла можно было бы выводить только выключением и снова включением микроконтроллера. Это абсолютно не требуется при использовании сторожевого таймера. Он может включаться и отключаться установкой битов конфигурации. Биты конфигурации могут устанавливаться только перед программированием.

Пример:

```

main                ;Главная программа
    bsf STATUS, RP0 ;Переключиться на Банк 1 (для выбора регистра)
    bcf STATUS, RP1
    movlw b'11001100' ;Установить коэффициент деления для предделителя
    movwf OPTION_REG ;сторожевого таймера WDT равный 1:16
    bcf STATUS, RP0 ;Переключиться на Банк 0
    bcf STATUS, RP1

new                ;Метка new
    movlw 0xFF       ;Загрузить в регистр W число 0xFF
    movwf 0x20        ;Переместить это число в регистр с адресом 0x20
wait              ;Метка wait
    decfsz 0x20       ;Уменьшить на 1 содержимое регистра 0x20 и
                    ;пропустить следующую команду, если результат = 0
    goto wait         ;Если регистр 0x20 не равен 0,
                    ;то перейти на метку wait
    clrwdt            ;Очистить сторожевой таймер WDT
    goto new         ;Перейти на метку new

```

Сторожевой таймер работает от внутреннего RC-генератора и поэтому не нуждается ни в каком внешнем тактовом генераторе. Типичное значение периода этого генератора для микроконтроллера PIC16F876A составляет 18 мс, но может быть изменено в пределах от 7 до 33 мс. Период этого генератора определяется значениями сопротивления внешнего резистора, емкостью внешнего конденсатора и рабочей температурой микроконтроллера. Таким образом без предварительного делителя частоты самое длительное время до сброса, при условии что сторожевой таймер не очищается, составляет около 18 мс. С помощью предварительного делителя частоты можно это время увеличить максимум до 2,3 секунд (максимальный коэффициент деления 1:128, поэтому $18 \text{ мс} \times 128 = 2304 \text{ мс} \approx 2,3 \text{ с}$). Коэффициент деления предварительного делителя частоты устанавливается в регистре OPTION_REG. В примере это значение устанавливалось на значение 1:16, а это значит, что процессор будет сбрасываться после 288 мс, если за это время не будет применена команда clrwdt. В примере осуществляется обратный счет в цикле снова и снова от значения 0xFF до 0x00.

Команда:	sleep
Назначение:	Перейти в режим сверхнизкого энергопотребления SLEEP
Синтаксис:	Sleep
Изменяемые флаги:	TO, PD

Микроконтроллеры часто применяются для устройств, питающихся от батареек. Поскольку батарейки имеют ограниченный срок эксплуатации, то надо как можно более экономно обходиться с находящейся в распоряжении энергией. Не требуется оставлять процессор постоянно включенным, например, если значение температуры должно измеряться всего только 2 секунды. Для перехода в режим сверхнизкого энергопотребления (*режим ожидания* — англ. *SLEEP*) и служит команда `sleep`. С помощью этой команды в рабочем состоянии поддерживается лишь часть микроконтроллера и поэтому-то и расходуется только минимум электроэнергии. Контроллер можно легко переключить назад в нормальное рабочее состояние различными способами, например, по переполнению сторожевого таймера WDT.

Пример:

```
bsf STATUS, RP0      ;Переключиться на Банк 1 (для выбранного регистра)
bcf STATUS, RP1
movlw b'11001111'    ;Установить коэффициент деления для предделителя
movwf OPTION_REG     ;сторожевого таймера WDT равный 1:128
bcf STATUS, RP0      ;Переключиться на Банк 0
bcf STATUS, RP1
read
movf PORTA, 0        ;Читать значение из порта А
movwf 0x20           ;Сохранить значение в регистр с адресом 0x20
clrwdt              ;Очистить сторожевой таймер WDT
sleep               ;Перейти в режим пониженного энергопотребления
goto read           ;Читать следующее значение после пробуждения
```

В примере значение из порта А читается приблизительно 2,3 секунды и сохраняется в регистре 0x20. После чтения порта процессор переходит снова в режим энергосбережения. Процессор может переводиться из состояния пониженного электропотребления в нормальный рабочий режим также по прерыванию, сменой сигнала на выводах порта В или прерыванием от периферии (например, USART).

3. Программирование с помощью MPLAB

После краткого рассмотрения структуры микроконтроллера в этой главе представляется *интегрированная среда проектирования MPLAB* от компании Microchip. Это профессиональный инструмент, в котором имеется все, что требуется для программирования микроконтроллеров. На веб-странице компании Microchip (www.microchip.com) можно бесплатно загрузить и использовать последние версии программного обеспечения. Одна из версий среды проектирования MPLAB находится также на прилагаемом к книге компакт-диске. В среде проектирования с помощью ассемблера можно программировать контроллеры, произведенные компанией Microchip. Конечно, могут быть использованы и другие встроенные С-компиляторы, для применения которых, однако, нужно приобретать у соответствующего производителя лицензию, что сопряжено с определенными материальными затратами.

Для программирования микроконтроллера в среде проектирования имеется текстовый редактор с синтаксической подсветкой (цветовое различение команд, чисел и примечаний). В текстовом редакторе могут задаваться *контрольные точки останова* (англ. *Breakpoints*), с помощью которых программный код можно выполнять частями, постепенно. Для просмотра текущего содержимого регистров можно выбирать между разными видами отображения. Кроме того, в среду проектирования входит встроенный симулятор. С его помощью можно писать программное обеспечение для микроконтроллера без подключения микроконтроллера. Персональный компьютер имитирует (симулирует) принцип работы контроллера. Можно посмотреть сигналы во времени, изменить состояние сигнала входных выводов или использовать секундомер, чтобы определить длительность цикла.

Чтобы отладить некоторую рабочую схему на основе микроконтроллера, аппаратные средства и программное обеспечение должны надлежащим образом между собой функционировать. При применении встроенного симулятора имеется очень хорошая возможность разделения программного обеспечения и аппаратных средств. В таком случае можно исследовать только программ-

ное обеспечение. Возможные ошибки монтажа микроконтроллера не играют тогда роли и программное обеспечение в микроконтроллер загружают только после успешной симуляции. Затем должны тестироваться возможности микроконтроллера и устраняться ошибки с помощью встроенного в среду проектирования отладчика. Таким образом, поиск и устранение ошибок может проходить в пошаговом режиме.

3.1. Установка MPLAB

Версия v8.10 интегрированной среды проектирования MPLAB находится в каталоге \Mplab\Installation на прилагаемом к книге компакт-диске. Установка программного обеспечения осуществляется после запуска файла Install_MPLAB_v810.exe.

3.2. Настройка каталога проекта

На компакт-диске находится много примеров, с которыми можно сразу же начать работать и имитировать со встроенным симулятором MPLAB. Чтобы работать с примерами, нужно скопировать их в папку на жестком диске, так как при изменениях в программном коде файлы также должны сохраняться. Во всех примерах была сохранена рабочая среда, так что нужные установки уже были сделаны при запуске. В этом случае программу можно запускать сразу. Разумеется, необходимо полностью скопировать каталог с примерами на жесткий диск, указав путь C:\PIC. В рабочей среде сохранено общее имя пути и поэтому при неверном указании имени каталога больше ничего не будет найдено. В таком случае в рабочей среде потребуется подробно указывать имя пути. Тем не менее, чтобы редактировать примеры, в другом каталоге можно выполнить настройки с помощью диалогового окна **Project Wizard** (Мастер проекта). Откройте пример двойным щелчком на файле с расширением msw. Затем откройте первое окно мастера проекта Project Wizard, используя команду меню **Project → Project Wizard** (Проект → Мастер проекта). В первых двух окнах мастера можно подтвердить все настройки по умолчанию, используя кнопку **Далее** (Next). Только в диалоговом окне на третьем шаге настройки **Step Three** (Третий шаг) (рис. 3.1) нужно выбрать раздел **Reconfigure Active Project** (Изменить конфигурации активного проекта) и в нем переключатель **Save changes to existing project file** (Сохранить изменения в существующем файле проекта).

После этого шага пути будут обновлены, и программа может быть скомпилирована. Если эта процедура не приводит к желаемому успеху, то нужно создать новый проект и импортировать необходимые файлы ассемблера. Как

создавать проекты и как может использоваться встроенный симулятор, объясняется далее.

Следующее описание объясняет только самые важные функции среды проектирования MPLAB, которые требуются для запуска примеров, данных в этой книге. Для более подробной информации на прилагаемом компакт-диске или веб-странице компании Microchip имеется полное руководство по эксплуатации (на английском языке), в котором достаточно подробно объяснены все свойства среды проектирования MPLAB.

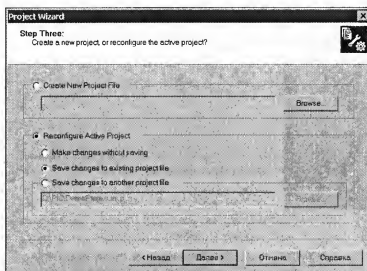


Рис. 3.1. Изменение конфигурации в диалоговом окне Project Wizard

3.3. Создание проекта

После успешной инсталляции среда проектирования MPLAB может запускаться. Чтобы написать программу на ассемблере для микроконтроллера, сначала нужно создать проект, в котором задается используемый микроконтроллер и имя проекта. Для создания проекта запустите мастер проекта **Project Wizard** (Мастер проекта) с помощью меню **Project → Project Wizard** (Проект → Мастер проекта) (рис. 3.2).

Откроется диалоговое окно **Project Wizard** (Мастер проекта) с приветствием (рис. 3.3). Для продолжения нажмите кнопку **Далее** (Next).

После этого откроется диалоговое окно (рис. 3.4), в котором с помощью раскрывающегося списка **Device** (Устройство) надо выбрать тип использу-

мого микроконтроллера. В нашем случае следует выбрать контроллер PIC16F876A.

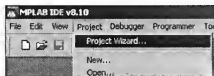


Рис. 3.2. Запуск мастера проекта Project Wizard

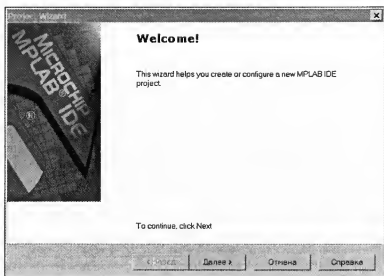


Рис. 3.3. Приветствие мастера проекта Project Wizard

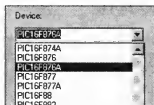


Рис. 3.4. Выбор микроконтроллера

После того как тип микроконтроллера был выбран, нужно установить, с помощью каких инструментов должна транслироваться написанная программа. Чтобы программировать на ассемблере, выбирают комплект инструментов **Microchip MPASM Toolsuite**. Где эти инструменты сохранены, уже задано

при установке среды проектирования MPLAB. Если инструменты не находятся, то путь можно указать с помощью кнопки **Browse** (Просмотр).

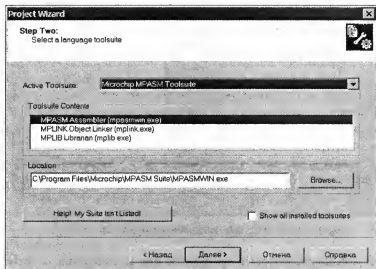


Рис. 3.5. Выбор инструмента

В следующем третьем окне мастера проекта есть возможность задать имя проекту и установить, в каком каталоге он должен сохраняться. Нажав кнопку **Browse** (Просмотр) в выбранной группе **Create New Project File** (Создать файл нового проекта) с помощью проводника можно выбрать соответствующую папку и в поле ввода задать имя файлу проекта (рис. 3.6).



Рис. 3.6. Указание пути для размещения нового проекта

Если имеются в наличии файлы из других проектов, то можно добавлять их к проекту с помощью следующего диалогового окна (рис. 3.7). Для этого нужно скопировать их заранее в каталог проекта, чтобы файлы проекта не были сохранены вразброс по всему жесткому диску. Впрочем, файлы можно добавить в проект и позже, поэтому это окно можно пройти без каких-либо указаний, нажав кнопку **Далее** (Next).

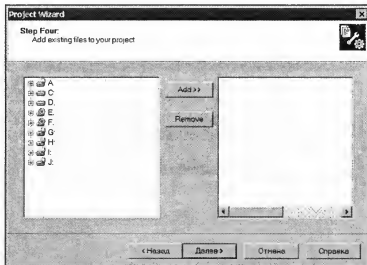


Рис. 3.7. Добавление файлов

Теперь все установки, которые требуются для размещения нового проекта, сделаны. В итоге получаем обзорное сводное окно мастера проектов, в котором показаны все выполненные установки (рис. 3.8). После их короткого контроля, если сделанные указания верны, можно щелкнуть на кнопке **Готово** (Finish), и проект будет создан, а для него на жестком диске будет отведена некоторая рабочая область.

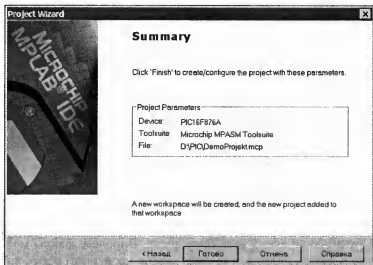


Рис. 3.8. Сводное окно мастера проектов

3.4. Рабочий стол MPLAB

На рабочем столе или иначе в рабочей области среды проектирования MPLAB пока еще ничего нет. Поэтому сначала нужно открыть два вспомогательных окна. С одной стороны, это — окно проекта. В нем будут показаны все файлы, которые относятся к проекту. Файлы для редактирования могут открываться двойным щелчком. Окно становится видимым, если в меню **View** (Вид) установить флажок перед пунктом **Project** (Проект) (рис. 3.9). Другое важное окно — это окно **Output** (Результат). В нем выводятся все сообщения, ошибки и предупреждения, которые возникают во время работы со средой проектирования MPLAB, поскольку важно знать, правильно ли подключены аппаратные средства и успешно ли завершился процесс преобразования. Это окно можно активизировать также в меню **View** (Вид), но при выборе пункта **Output** (Результат). Теперь можно настраивать размеры окон и их расположение на рабочей области. При завершении работы со средой проектирования MPLAB это расположение окон сохраняется, так что можно стартовать после обновленного вызова проекта снова и снова с последнего его представления.

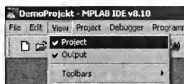


Рис. 3.9. Настройка вида проекта

Поскольку при создании проекта никакие файлы в проект не включались, теперь нужно создать новый файл на ассемблере. Для этого из меню **File** (Файл) выберите пункт **New** (Новый). В результате откроется окно с именем **Untitled** (Без названия) (рис. 3.10). Символ * (звездочка), стоящий после имени окна, показывает, что в этом файле производились некоторые изменения, но они еще пока не сохранены. В этом окне текстового файла можно вводить и редактировать код программы на ассемблере.



Рис. 3.10. Создание нового файла

Чтобы сохранить текстовый файл после редактирования, нужно в меню **File** (Файл) выбрать пункт **Save As** (Сохранить как). Затем выбрать желаемый путь и дать файлу имя (рис. 3.11). Важно, что за именем файла указывается его расширение, которое указывает, к какому типу относится тот или иной файл. Файлы, написанные на ассемблере, содержат расширение `asm`. Для файлов, в которых определяются константы или макрокоманды, используется расширение `inc` (от англ. *Include* — включить, присоединить). После того как файл будет записан с именем и расширением, в файле при его отображении начнет функционировать синтаксическая подсветка, т. е. для комментария, чисел и команд кода (например, `movlw`) используются разные цвета. Для небольшого упрощения работы при создании файла на ассемблере на компакт-диске можно найти шаблон, в котором уже содержится структура файла, созданного на ассемблере. Здесь уже имеются биты конфигурации и установлен необходимый код для программы обработки прерывания. Шаблон находится в файле `\Beispiele\Vorlage.asm`.

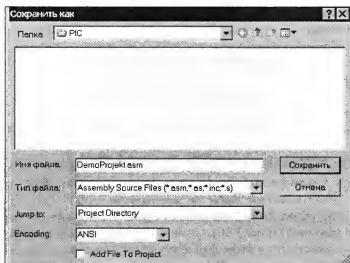


Рис. 3.11. Сохранение файла проекта

Файл с кодом на ассемблере теперь будет сохранен на жестком диске. Разумеется, он пока не будет находиться в проекте и поэтому должен быть добавлен в проект. Для этого в меню **Project** (Проект) щелкните пункт **Add Files to Project** (Добавить файлы в проект) (рис. 3.12) и в открывшемся диалоговом окне выберите желаемый файл. Таким же способом можно добавлять к проекту другие уже созданные файлы. Файлы перед добавлением в проект должны находиться в каталоге проекта, чтобы все редактируемые файлы были размещены в общем каталоге.

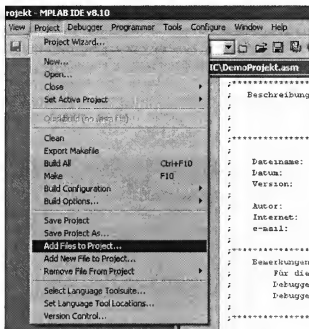


Рис. 3.12. Добавление файлов к проекту

После того как файл на ассемблере будет успешно вставлен в проект, он также появится в окне проекта (рис. 3.13). Обратите внимание на то, что за именем папки проекта стоит звездочка, которая указывает на то, что текущие изменения еще не сохранены.

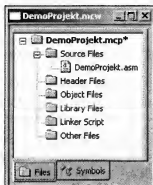


Рис. 3.13. Отображение окна проекта

Если код программы введен и соответствующий файл сохранен, то нужно протестировать программу в режиме реального времени и устранить возмож-

ные ее ошибки. Интегрированная среда проектирования MPLAB располагает встроенной программой-симулятором, с помощью которой может имитироваться работа PIC-контроллера. Поэтому вовсе не требуется, чтобы микроконтроллер непосредственно присоединялся к компьютеру. Чтобы использовать встроенный *симулятор* для отладки программы, выберите в меню **Debugger** → **Select Tool** (Отладчик → Выбор инструмента) пункт **MPLAB SIM** (Симулятор MPLAB SIM). Отметка у этого пункта показывает на активизацию инструмента отладки (рис. 3.14).

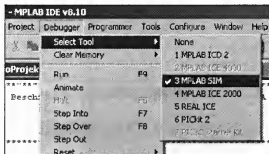


Рис. 3.14. Выбор инструмента для отладки

После выбора симулятора инструментальной панели появляются дополнительные значки кнопок (рис. 3.15). С помощью этих кнопок можно управлять отладкой и постепенно устранять ошибки в программе.

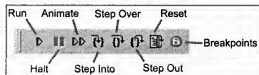


Рис. 3.15. Кнопки для симуляции

С помощью этих кнопок можно постепенно пройти программу и таким образом проверить воздействие каждой отдельной команды. Следующее описание поясняет принцип действия команд, выполняемых с помощью этих кнопок:

- **Run** (Выполнить). Этой командой запускается выполнение программы. Команды программы отрабатываются по очереди, пока не будет останова. Выполнение может прерываться кнопкой **Halt** (Останов) или в точке останова.
- **Halt** (Останов). Команда вызывает прерывание работающей программы.
- **Animate** (Анимация). Симулируется выполнение программы, при этом выполняются действия, аналогичные пошаговому режиму, обновляя зна-

чения регистров после каждой инструкции. При выполнении этой команды можно увидеть, на каком месте в данный момент находится программа, а также остановить ее выполнение по требованию.

- ❑ **Step Into** (Подробный пошаговый режим). При каждом щелчке по этой кнопке запускается одна команда программы. Если появляется вызов подпрограммы при выполнении программы, то она также запускается в пошаговом режиме.
- ❑ **Step Over** (Пошаговый режим с обходом). Любая команда программы запускается пошагово как при нажатии кнопки **Step Into**, разумеется, вызов подпрограммы осуществляется за один шаг, но пошагового редактирования подпрограммы при этом не будет.
- ❑ **Step Out** (Выход из пошагового режима). Если перешли на подпрограмму в пошаговом режиме выполнения, то с помощью этой команды можно завершить выполнение всех команд в подпрограмме и снова вернуться в основную программу.
- ❑ **Reset** (Сброс). Щелчок по этой кнопке вызывает сброс процессора, и программа запускается сначала.
- ❑ **Breakpoints** (Точки останова). С помощью данной кнопки получают обзор установленных точек останова, кроме того, их можно добавить или удалить.

Перед выполнением программы она должна быть обязательно скомпилирована. Чтобы сделать программу способную к выполнению, в меню **Project** (Проект) выберите пункт **Build All** (Компилировать все исходные файлы) (рис. 3.16). Аналогичное действие происходит, если нажать кнопку для выполнения программы (**Run, Step Into** (Выполнить, Подробный пошаговый режим)). Если были изменения после последнего выполнения программы, то программный код автоматически по-новому компилируется.

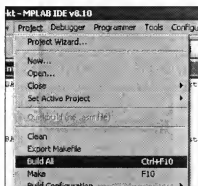


Рис. 3.16. Компиляция проекта

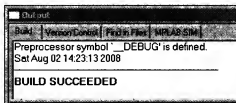


Рис. 3.17. Сообщение об успешной компиляции программы в окне Output

Если при программировании не было ошибок и компиляция прошла успешно, то в окне **Output** (Результат) появится сообщение BUILD SUCCEEDED (Компиляция прошла успешно) (рис. 3.17). Если проявилась ошибка, то ее можно локализовать двойным щелчком на сообщении об ошибке, а затем ее устранить.

Теперь программа готова к выполнению и может подробно анализироваться.

3.5. Меню View

Для анализа программы в распоряжении пользователя находятся различные утилиты. Они могут выбираться с помощью меню **View** (Вид) (рис. 3.18) путем выбора соответствующего пункта.

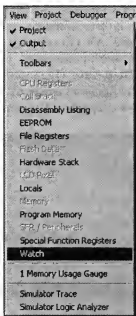
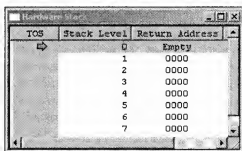


Рис. 3.18. Меню View

3.5.1. Аппаратный стек

При использовании в программе программ обработки прерывания и подпрограмм имеется возможность просматривать состояние стека. Зеленая стрелка в столбце **TOS** (Вершина стека) показывает текущий уровень, а в столбце **Return Address** (Адреса возврата) стоят сохраненные адреса возврата (рис. 3.19).



TOS	Stack Level	Return Address
⇒	0	Empty
	1	0000
	2	0000
	3	0000
	4	0000
	5	0000
	6	0000
	7	0000

Рис. 3.19. Аппаратный стек

3.5.2. Окно наблюдения

Очень важное окно — окно **Watch** (Наблюдение) (рис. 3.20), предназначенное для наблюдения за переменными. С помощью этого окна можно посмотреть содержимое всех регистров и также изменить их во время останова. Если же содержимое регистра изменяется во время выполнения программы, то его значение для привлечения внимания отображается красным цветом.

Все регистры разделены на 2 группы.

Одна группа — это регистры специального назначения (PCH) — **Special Function Register (SFR)**, другая — это самостоятельно определенные регистры общего назначения (POH) со своим символическим именем. К регистрам специального назначения относятся в том числе также регистр **W** или регистры портов (например, **PORTA**). Чтобы наблюдать за новым регистром, сначала, в левом верхнем открывающемся списке, выберите соответствующий регистр и добавьте его к списку отображаемых с помощью щелчка на кнопке **Add SFR** (Добавить PCH) или **Add Symbol** (Добавить символ). Кроме того, можно непосредственно в список добавить регистр с определенным адресом (например, 020). Для этого выберите нижнюю строку и укажите в столбце **Address** (Адреса) адрес в шестнадцатеричном формате (без 0x). При помощи мыши способом перетаскивания можно изменять последовательность регистров. Нажимая на закладки от **Watch 1** до **Watch 4**, расположенные в левом

нижнем углу окна **Watch** (Наблюдение), можно делать различные сочетания наблюдаемых переменных.

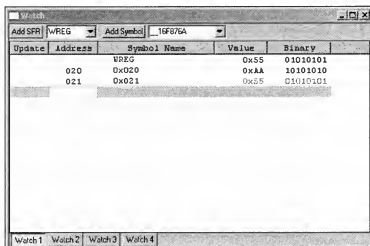


Рис. 3.20. Окно Watch

3.5.3. Листинг дизассемблера

Иногда необходимо знать, как транслирован некоторый программный код и что загружается, в конечном счете, конкретно в микроконтроллер. Для этого имеется *дизассемблер*, который сгенерированный машинный код снова преобразует в более понятный код на ассемблере. К сожалению, при этом все ранее определенные имена регистров пропадают и представляются только лишь адресом. В листинге, полученном с помощью дизассемблера (рис. 3.21), в левом столбце можно увидеть адреса программного кода, а правее,

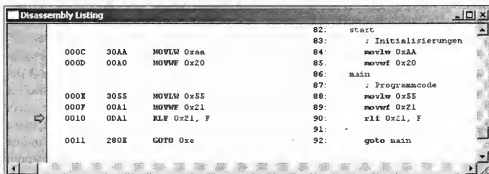


Рис. 3.21. Окно Disassembly Listing

рядом с ним, машинный код. В третьем столбце находится обратно оттранслированный код на ассемблере. При наличии программного кода на ассемблере он располагается в правой части листинга и выглядит так, как он и программировался. Зеленая стрелка слева показывает текущее состояние программного счетчика. Если не понятно, на каком шаге выполнения находится программа, всегда для уточнения можно просмотреть листинг дизассемблера, потому что именно этот код записан непосредственно в микроконтроллер.

3.5.4. EEPROM-память

Поскольку многие микроконтроллеры содержат внутреннюю EEPROM-память, в среде проектирования MPLAB имеется возможность просматривать данные, которые в ней хранятся. В окне **EEPROM** (ЭСПЗУ), в правой его части, можно увидеть интерпретацию данных, представленную в ASCII-формате (рис. 3.22). Если, например, требуется найти содержимое ячейки памяти с адресом 0x93, то нужно сверху вниз просмотреть левый столбец **Address** (Адрес) вплоть до строки 90 и далее в этой строке выбрать значение в столбце **03**. Именно здесь, на пересечении указанных строки и столбца, и будет находиться содержимое ячейки с адресом 0x93.

Address	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	ASCII
00	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
10	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
20	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
30	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
40	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
50	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
60	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
70	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
80	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
90	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
93	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
A0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
B0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
C0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
D0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
E0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
F0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF

Рис. 3.22. Окно EEPROM

3.6. Точки останова

При программировании часто требуется тестировать и просматривать определенные места кода. Чтобы с помощью команд **Step Into** (Подробный пошаговый режим) или **Step Over** (Пошаговый режим с обходом) последовательно не выполнять всю программу до нужного места программы, устанавливают

точку останова перед требуемой частью кода. Далее можно запустить программу при помощи команды **Run** (Выполнить), тогда отладчик (Debugger) последовательно выполнит все команды и остановится на выбранном месте. Следует иметь в виду, что для установки точек останова существуют различные возможности. Можно, например, щелкнуть на кнопке **Breakpoints** (Точки останова) и затем указать нужную позицию или же, в простейшем случае, сделать двойной щелчок на команде в желаемой строке. При этом активная точка останова будет обозначена красным кругом с символом "B" белого цвета (рис. 3.23). Программа после запуска остановится на этом месте, а текущие значения регистров будут показаны в окне наблюдения за переменными **Watch** (Наблюдение). Далее можно выполнять программный код в пошаговом режиме и просматривать изменение регистров после каждого шага. С помощью симулятора можно установить сколь угодно точек останова, в то же время, как, в частности, в настоящем микроконтроллере, из-за ограниченных ресурсов в большинстве случаев возможна только одна точка останова. Однако если нужно иметь несколько точек останова в программе, то после первого останова надо удалить текущую точку останова и установить ее на новое необходимое место.

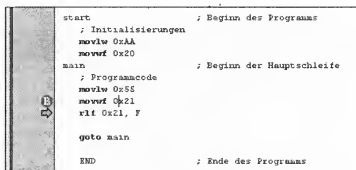


Рис. 3.23. Точки останова

3.7. Симулятор

С помощью описанных ранее возможностей можно просматривать внутренние регистры и интерпретировать их изменение. Поскольку микроконтроллер располагает, как правило, и входами, то и они также должны иметь возможность для симулирования, т. е. симуляции внешних воздействующих тестирующих сигналов, иначе *стимулов*. Для осуществления симуляции, выберите из меню **Debugger** (Отладчик) пункт **Stimulus** (Стимул), а затем **New Workbook** (Новая рабочая книга). В результате откроется новое окно **Stimulus** (Стимул) с несколькими различными вкладками.

3.7.1. Основные настройки

Перед началом работы с симулятором, нужно выполнить еще несколько основных настроек. Окно для задания параметров можно открыть с помощью меню **Debugger** (Отладчик) и пункта **Settings** (Параметры). Самый важный параметр для настройки находится на вкладке **Osc/Trace** (Генератор/Трассировка). Здесь должна устанавливаться требуемая тактовая частота микроконтроллера, чтобы внешние сигналы могли подаваться в нужный момент (рис. 3.24). На всех других вкладках параметры можно оставить по умолчанию.

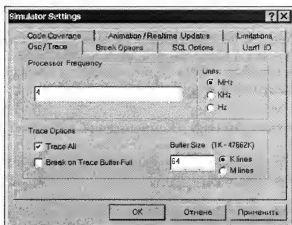


Рис. 3.24. Окно Simulator Settings

3.7.2. Асинхронный стимул

На вкладке **Asynch** (Асинхронный) в окне **Stimulus** (Стимул) при выполнении программы есть возможность на входной вывод микроконтроллера подавать заранее определенное значение сигнала (рис. 3.25). Асинхронно значит, что состояние вывода может изменяться в любой момент. После выбора желаемого вывода и описания того или иного действия его активизация осуществляется с помощью кнопки с символом ">", расположенной в первом левом столбце **Fire** (Запустить). Если во время выполнения программы нажать на эту кнопку, то запускается действие, которое стоит в этой строке. Во втором столбце **Pin/SFR** (Вывод/РЧН) можно выбрать соответствующий вывод контроллера, а затем в столбце **Action** (Действие) выбрать любое из 5 следующих различных действий.

- ☐ **Toggle** (Переключать). При выполнении этого действия состояние сигнала на соответствующем выводе меняется на противоположное. Если на выво-

де был высокий логический уровень, то после нажатия этой кнопки становится низкий логический уровень и наоборот.

- ☐ **Set High** (Установить высокий уровень). На вход подается высокий логический уровень, независимо от уровня, приложенного ранее.
- ☐ **Set Low** (Установить низкий уровень). На вход подается низкий логический уровень.
- ☐ **Pulse High** (Импульс высокого уровня). На вывод подается импульс высокого логического уровня и определенной длительности. Длительность импульса задается в столбцах **Width** (Длительность) и **Units** (Единица измерения). После завершения импульса на вывод снова подается низкий логический уровень.
- ☐ **Pulse Low** (Импульс низкого уровня). На вывод подается импульс низкого логического уровня и определенной длительности. Длительность импульса задается в столбцах **Width** (Длительность) и **Units** (Единица измерения). После завершения импульса на вывод снова подается высокий логический уровень.

В столбец **Comments/Message** (Комментарии/Сообщение) можно вносить примечания, которые предназначены для пояснения выполняемого действия. Любое действие может запускаться в пошаговом режиме выполнения или во время выполнения программы.

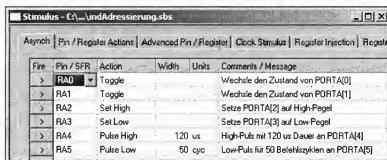


Рис. 3.25. Настройка асинхронного стимула

3.7.3. Циклический синхронный стимул

Для стимулирования входных выводов в автоматическом режиме требуется перейти на вкладку **Pin/Register Actions** (Вывод/Действия регистра) (рис. 3.26). На этой вкладке можно задать, в какой момент времени на определенном выводе должен появиться высокий или низкий логический уровень сигнала. В столбце **Time** (Время) указывается момент времени, в который

должна происходить смена состояния сигнала, а в столбцах справа от столбца **Time** (Время) можно установить уровни сигналов, которые должны быть приложены в этот момент. Чтобы добавить столбцы выводов для задания сигналов в таблицу, надо щелкнуть на верхней строке **Click here to Add Signals** (Щелкнуть здесь для добавления сигналов).

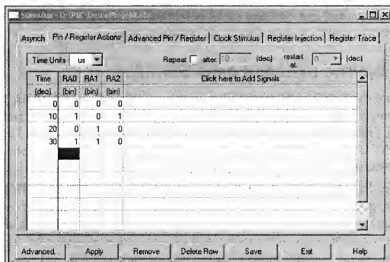


Рис. 3.26. Настройка синхронного стимула

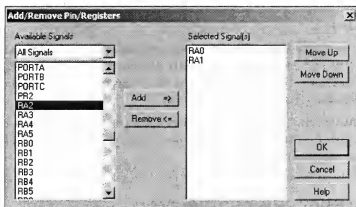


Рис. 3.27. Добавление выводов для задания сигналов

В результате будет открыто следующее окно (рис. 3.27), в котором можно выбрать нужные выходы, регистры специального назначения или битовые поля. В левом списке **Available Signals** (Доступные сигналы) расположены

все доступные выводы для сигналов, которые могут быть добавлены в таблицу после нажатия на кнопку **Add** (Добавить) в список выбранных выводов для сигналов (**Selected Signal(s)**). После нажатия кнопки **Remove** (Удалить) выводы могут быть удалены из списка. Последовательность выводов в списке можно изменить с помощью кнопок **Move Up** (Переместить вверх) и **Move Down** (Переместить вниз). Кнопка **Move Up** (Переместить вверх) перемещает выбранный вывод для сигнала на одну позицию вверх, а кнопка **Move Down** (Переместить вниз) — перемещает на позицию вниз.

3.7.4. Другие вкладки окна Stimulus

Вкладки **Advanced Pin/Register** (Расширенные возможности), **Clock Stimulus** (Синхронизация стимула), **Register Injection** (Ввод данных в регистр) и **Register Trace** (Трассировка регистра) предназначены для более тонкой настройки и для понимания основ работы большого значения не имеют, а поэтому детально здесь не рассматриваются. Следует заметить, что подробные сведения могут быть взяты из документации на среду проектирования MPLAB.

На вкладке **Advanced Pin / Register** (Расширенные возможности) могут определяться расширенные возможности по синхронизации моментов подачи тестовых сигналов на указанные выводы или регистры в зависимости от заявленных условий.

Вкладка **Clock Stimulus** (Синхронизация стимула) может использоваться для определения момента времени начала и конца подачи тактовых сигналов на указанный вывод.

На вкладке **Register Injection** (Ввод данных в регистр) имеется возможность загружать значения в регистр. Значения должны быть заранее определены с помощью текстового файла, который можно указать на этой вкладке в столбце **Data Filename** (Имя файла с данными).

Чтобы отслеживать изменение содержимого регистров, можно с помощью вкладки **Register Trace** (Трассировка регистра) указывать имя файла, в котором должны сохраняться данные.

3.8. Логический анализатор

Поскольку анализировать сигналы в зависимости от времени только при помощи изменений регистра достаточно трудно, то можно применить специальный логический анализатор, активизировав одноименное окно **Logic Analyzer** (Логический анализатор) (рис. 3.28).

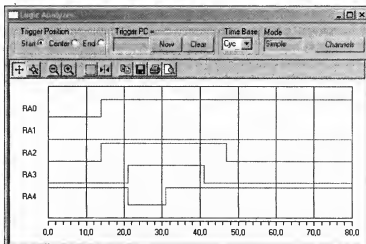


Рис. 3.28. Окно Logic Analyzer

Чтобы его открыть, нужно в меню **View** (Вид) выбрать пункт **Simulator Logic Analyzer** (Логический анализатор симулятора). Прежде чем сигналы начнут отображаться, нужно выбрать их с помощью кнопки **Channels** (Каналы). После этого откроется диалоговое окно для добавления или удаления сигналов, которое можно выполнить воспользовавшись соответствующими кнопками **Add** (Добавить) и **Remove** (Удалить) (рис. 3.29).

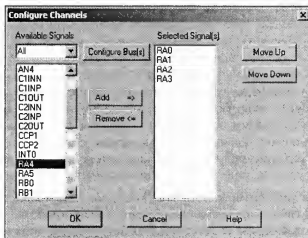


Рис. 3.29. Выбор каналов логического анализатора

После того, как нужные сигналы будут выбраны, они появятся в окне **Logic Analyzer** (Логический анализатор) (см. рис. 3.28). Для более детального

отслеживания изменения сигналов нужно выполнять программу в пошаговом режиме. Если же программе позволяют свободно выполняться, то количество записанных циклов очень быстро вырастет и при этом будет очень трудно или вовсе не возможно определить в представленном изображении короткий импульс. Однако программу можно запускать до точки останова и затем исследовать представленные значения более подробно. Для увеличения или уменьшения масштаба изображения можно в окне **Logic Analyzer** (Логический анализатор) воспользоваться кнопками с лупой.

3.9. Внутрисхемный отладчик ICD 2

Поскольку хотелось бы не только моделировать микроконтроллер, но и использовать его в схеме, требуется загружать написанную программу в память микроконтроллера. Для этого имеется множество вариантов. Можно программировать контроллер с помощью программаторов самых различных производителей. Многие устройства позволяют только просто программирование. Однако существуют самые различные устройства, при помощи которых микроконтроллер может быть отлажен, в то время как он находится в собранной схеме. Электрическую схему простейшего программатора можно найти на прилагаемом компакт-диске. При этом программировать можно различные PIC-микроконтроллеры, но не все из них можно будет отладить. Тем не менее, схему для мини-программатора можно собрать из малого количества комплектующих изделий, которые наверняка имеются в наличии почти у каждого любителя мастерить. Из Интернета со страницы www.qsl.net/dl4yhf для программирования микроконтроллеров можно загружать соответствующее бесплатное программное обеспечение. При применении его, однако, обратите внимание на соблюдение авторских прав. Если программное обеспечение вполне может отлаживаться с помощью симулятора, то такой простой малозатратный вариант вполне подойдет. Если же программы получаются очень большими и поиск ошибок должен осуществляться с реальными сигналами, то при этом можно быстро натолкнуться на проблему. А ограничение простого программатора состоит в том, что не все PIC-микроконтроллеры могут программироваться при помощи такого программатора.

Чтобы использовать возможности микроконтроллера полностью, требуется покупать эффективное программное и отладочное устройство. Следует заметить, что в Интернете кроме этого имеется много радиолюбительских проектов программаторов, с помощью которых могут программироваться PIC-контроллеры. Если же остановить свой выбор на инструменте ICD 2 от компании Microchip, то можно получить преимущество, которое связано с тем, что программатор непосредственно может использоваться из среды проекти-

рования MPLAB и никакое дополнительное программное обеспечение для программирования не потребуется. Чтобы посмотреть, какие микроконтроллеры могут быть запрограммированы и каким программатором, можно воспользоваться диалоговым окном выбора устройств **Select Device** (Выбор устройства) (рис. 3.30), которое открывается из меню **Configure** (Конфигурация) после выбора одноименного пункта **Select Device** (Выбор устройства). В этом окне индикаторы зеленого цвета означают, что выбранный микроконтроллер полностью поддерживается, а желтого цвета — что устройство протестировано не полностью и находится еще в фазе совершенствования. Инструменты, отмеченные индикаторами красного цвета, не функционируют с выбранным микроконтроллером.

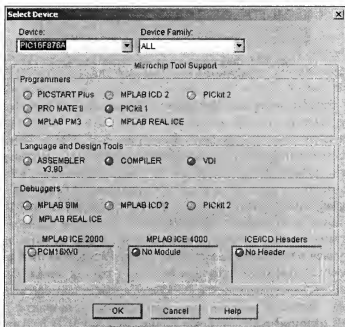


Рис. 3.30. Выбор микроконтроллера

Далее объясняется программирование и анализ микроконтроллера с *внутри-схемным отладчиком ICD 2 (In-Circuit-Debugger)*. Он нашел большое распространение и используется многими разработчиками.

После инсталляции отладчик можно выбрать из меню **Debugger** → **Select Tool** (Отладчик → Выбор инструмента) щелчком на пункте **MPLAB ICD 2** (рис. 3.31).

Если был выбран этот пункт меню, но не был присоединен микроконтроллер, то появится предупреждение "Invalid Target Device Id" (Не правильный иден-

тификатор целевого устройства). В данный момент это сообщение не так уж важно и может быть проигнорировано. Однако, чтобы проблемы не возникали, прежде чем подключить отладчик ICD 2 к микроконтроллеру, нужно выполнить правильные настройки. Для этого сначала нужно запустить мастер настроек MPLAB ICD 2.



Рис. 3.31. Выбор в качестве инструмента отладки внутрисхемного отладчика ICD 2

Это можно сделать из меню **Debugger** (Отладчик), выбрав пункт **MPLAB ICD 2 Setup Wizard** (Мастер настроек MPLAB ICD 2). Мастер предназначен для задания начальных основных параметров. Здесь, в первом диалоговом окне, в качестве интерфейса связи выбирают порт USB. В следующем окне можно задать способ подачи питания на микроконтроллер. Если требуется тестировать микроконтроллер в схеме с незначительным потреблением тока, то можно использовать переключатель **Power target from the MPLAB ICD 2** (Подача питания к объекту от MPLAB ICD 2). В таком случае надо обеспечить отсутствие подачи внешних напряжений на микроконтроллер. В большинстве случаев надо выбрать переключатель **Target has own power supply** (Объект имеет собственный источник питания), так как таким образом можно использовать полную электрическую схему аппаратных средств с источником питания. Затем, в следующих двух окнах, установите два флажка: **MPLAB IDE automatically connects to the MPLAB ICD 2** (Автоматическое подключение MPLAB IDE к MPLAB ICD 2) и **MPLAB ICD 2 automatically downloads the required operating system** (Автоматическая загрузка MPLAB ICD 2 необходимой операционной системы). Эти оба параметра облегчают работу с отладчиком, поскольку среда проектирования MPLAB автоматически подключается к отладчику и загружается необходимое программное обеспечение. В последнем итоговом окне после просмотра всех настроек можно их принять, нажав кнопку **Готово** (Finish).

Теперь в схему и на микроконтроллер можно подать напряжение и сформировать подключение с помощью пункта **Connect** (Подключение) в меню **Debugger** (Отладчик). Если все подключения и установки были сделаны пра-

вильно, то в окне **Output** (Результат) (рис. 3.32) будет сообщение об успешном подключении.

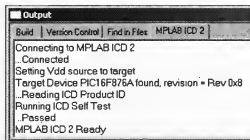


Рис. 3.32. Информация в окне Output

Если же при подключении все же будут проблемы, то можно еще раз посмотреть и исправить установки, сделанные при помощи мастера настроек MPLAB ICD 2, воспользовавшись пунктом **Settings** (Параметры) из меню **Debugger** (Отладчик). На вкладке **Status** (Состояние) можно увидеть, подключен ли отладчик ICD 2 и пройдено ли самотестирование (**Self Test**).

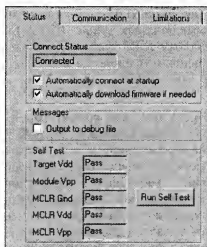


Рис. 3.33. Состояние отладчика ICD 2

С помощью вкладки **Power** (Питание) можно проверить, находятся ли приложенные напряжения в нужном диапазоне. Для микроконтроллера с напряжением питания 5 В показанное целевое напряжение V_{DD} должно составлять 4,5—5,5 В. Напряжение программирования V_{PP} должно быть между 9,0 и 13,5 В. Оно предоставляется в распоряжение программатором. С помощью

кнопки **Update** (Обновление) можно обновлять отображение значений напряжения или измерений.

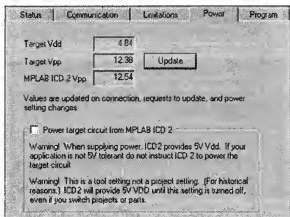


Рис. 3.34. Источник питания

Следующая интересная вкладка — это вкладка **Program** (Программирование) (рис. 3.35). Здесь можно устанавливать, какая область памяти должна записываться при программировании. В большинстве случаев вполне достаточно, когда активизацией переключателя **Allow ICD 2 to select memories and ranges** (Разрешить отладчику ICD2 выбор памяти и диапазона) позволяют отладчику ICD 2 устанавливать параметры самостоятельно. Только в таком случае во Flash-память записывается такой объем, который требуется для программы.

В конце можно просмотреть еще биты конфигурации. Обзор текущих значений можно получить из меню **Configure** (Конфигурация), выбрав пункт **Configuration Bits** (Биты конфигурации). Если микроконтроллер работает не так как требуется, то в первую очередь нужно проверить эти установки и настроить их в соответствии с требованиями. При установленном флажке, расположенном в верхней строке, в окне отображаются биты конфигурации, которые установлены в исходном программном коде. Чтобы сделать какие-либо изменения настроек, этот флажок должен быть снят. Для поиска ошибок или отладки настройки должны быть такие, как это показано на рис. 3.36.

Следующие настройки в отношении отладки и программирования могут осуществляться с помощью пункта **Settings** (Параметры) в меню **Configure** (Конфигурация). Чтобы быть уверенным, что изменения сохраняются перед запуском программ на жестком диске, в открывшемся окне на вкладке **Debugger** (Отладчик) нужно установить флажок **Automatically save files before running** (Автоматическое сохранение файлов перед выполнением) (рис. 3.37).

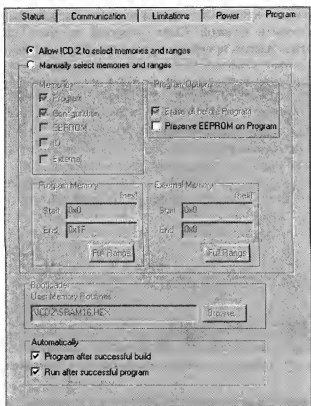


Рис. 3.35. Настройки для программирования

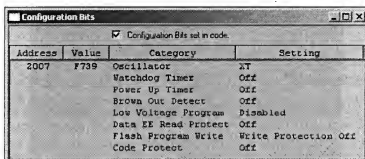


Рис. 3.36. Биты конфигурации

Для указания памяти, которая должна очищаться перед программированием чипа, нужно перейти на вкладку **Program Loading** (Загрузка программы). Для того чтобы при программировании быть наверняка уверенным, что во

Flash-памяти никаких кусков от старых программ больше нет, на этой вкладке нужно установить флажок **Clear program memory upon loading a program** (Очистить память программ при загрузке программы) (рис. 3.38).

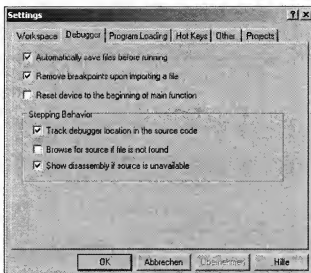


Рис. 3.37. Параметры отладки на вкладке **Debugger** в окне **Settings**

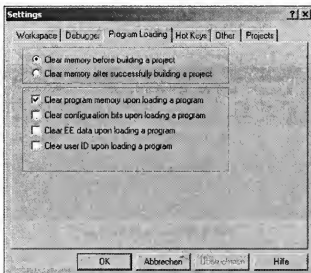


Рис. 3.38. Настройки для программирования

3.10. Программирование

Если после устранения всех ошибок в программе ее необходимо загрузить во Flash-память, внутрисхемный отладчик ICD 2 должен быть установлен в режим программирования. Для этого в меню **Programmer** (Программатор) выберите пункт **Select Programmer** (Выбор программатора), а затем **MPLAB ICD 2** (рис. 3.39). Дело в том, что внутрисхемный отладчик ICD 2 не может функционировать одновременно в режиме отладки и программирования и поэтому должен переключаться. Если микроконтроллер применяется в готовой схеме, которая предназначена для продажи, то большинство разработчиков заинтересовано в том, чтобы программный код не смог быть прочитан конкурентами. Поэтому перед программированием нужно соответствующим образом установить биты конфигурации. В частности, для защиты программного кода от чтения бит конфигурации **Code Protect** (Код защиты) должен иметь значение **ON** (Включено).

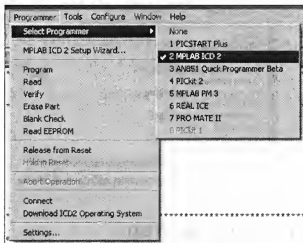


Рис. 3.39. Использование отладчика ICD 2 в качестве программатора

После того как выбрали встроенный отладчик ICD 2 в качестве *программатора*, осуществляется подключение между средой проектирования MPLAB и отладчиком ICD2, идет поиск присоединенного микроконтроллера и выполняется самотестирование. Прошло ли подключение безупречно, будет известно из сообщения в окне **Output** (Результат) (рис. 3.40). Если нет сообщений об ошибке при установке связи, то чип может программироваться с помощью пункта меню **Programmer** → **Program** (Программатор → Программирование). Появляющиеся сообщения информируют о действии, которое только что запускалось. Сначала проверяются биты конфигурации, очищается па-

мать и новая программа пишется во Flash-память. Если программа была записана в память, то в конце еще раз проверяется, правильно ли записались все данные. Потом проверенные биты конфигурации пишутся в микроконтроллер. В конце после вывода сообщения "Programming Succeeded" (Программирование прошло успешно) (см. рис. 3.40) микроконтроллер готов к применению и может использоваться для окончательной схемы.

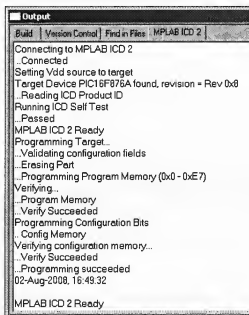


Рис. 3.40. Сообщение об успешном программировании

3.11. Текстовый редактор

Все примеры программ в этой книге были написаны с помощью встроенного в среду проектирования MPLAB текстового редактора. Если при открытии файла команды не выглядят равномерно расположенными сверху вниз, то значение для шага табуляции установлено, вероятно, ошибочно. Чтобы получить наглядное изображение программного кода, нужно в меню **Edit** (Редактировать) выбрать пункт **Properties** (Свойства) и затем в открывшемся окне на вкладке 'ASM' **File Types** (Типы файла ASM) в поле **Tab size** (Размер табуляции) установить значение, равное 3. На этой вкладке в файле, написанном на ассемблере, можно также включать или отключать нумерацию строк. Кроме того, здесь можно задать установку точек останова в программе посредством двойного щелчка на желаемой строке.

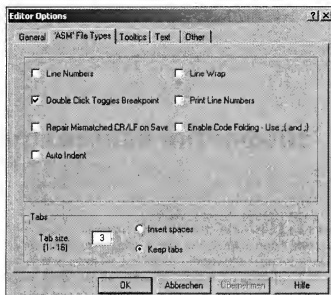


Рис. 3.41. Настройка параметров текстового редактора

4. Программный интерфейс

В предыдущих главах уже были рассмотрены некоторые основы программирования. В этой главе речь пойдет о том, как подключить микроконтроллер к программатору и как должна выглядеть окончательная схема, чтобы написанная программа функционировала в готовых аппаратных средствах.

Для загрузки программы в PIC-контроллер предоставляются в распоряжение 3—4 вывода. Они могут использоваться после программирования по требованию также для других целей, например как вход или выход. Здесь речь идет о следующих выводах:

- PGC (Programming Clock) — тактовый вход в режиме программирования;
- PGD (Programming Data) — вывод для данных в режиме программирования;
- V_{PP} (Programming Voltage) — входной вывод для высокого напряжения программирования, примерно 11—13 В;
- PGM (Low-Voltage Programming Enable) — входной вывод для разрешения режима низковольтного программирования PIC-контроллера.

Наряду с этими выводами требуются еще два вывода для напряжения питания (V_{DD} и V_{SS}). Если микроконтроллер не подключен к схеме, в которой имеется рабочее напряжение, то для программирования должно предоставляться в распоряжение напряжение питания от программатора.

4.1. Программирование с помощью ICD 2

Далее объясняется программирование PIC-контроллера с помощью выводов PGC, PGD и V_{PP} при помощи встроенного отладчика ICD 2. На прилагаемом к книге компакт-диске находится короткое описание, как можно программировать микроконтроллер с помощью небольшой схемы, изготовленной самостоятельно. Схема собственного изготовления со всеми комплектующими изделиями стоит менее 2 € (≈80 руб.). К сожалению, с помощью этой схемы

нельзя отлаживать контроллеры, в схеме и таким образом проверять интерфейсы аппаратных средств. Разумеется, такой схемы вполне достаточно, если программа успешно составлена и смоделирована, например программа "бегущего огня", и ее надо лишь загрузить в PIC-контроллер. Поскольку схема состоит из малого количества простых комплектующих деталей и поэтому может быть быстро собрана на специальной монтажной плате с отверстиями. При монтаже схемы нужно стараться делать как можно более короткие соединения с микроконтроллером, поскольку имеется возможность возникновения помех, и тогда микроконтроллер будет запрограммирован неправильно. Нужно понимать, что от этой простой схемы нельзя требовать того же, что и от полнофункционального профессионального программного и отладочного устройства. Если хотите более детально разобраться с программированием разных типов контроллеров, то можно взять подробное описание из руководства по эксплуатации отладчика ICD 2. А так же, если хотите знать больше о программировании Flash-памяти, ознакомьтесь с документом FLASH Memory Programming Specification PIC16F87XA (файл на прилагаемом компакт-диске FlashMemoryProgramming_Spec_PIC16F876A.pdf).

Чтобы избежать неприятностей при программировании и отладке, нужно использовать по возможности выводы PGC и PGD исключительно для целей программирования. Если все же требуется по каким-либо причинам использовать эти выводы для других целей, то нужно стараться определять для этих выводов менее важные функции (например, предупреждение, если напряжение батареи не соответствует определенному значению) или присоединять простые пассивные схемы (например, кнопки). В худшем случае отладка микроконтроллера в схеме будет не возможна. Вывод PGM не является необходимым для программирования, однако здесь нужно обращать внимание на некоторые моменты. С помощью этого вывода микроконтроллеру сообщается, что реализуется режим низковольтного программирования с пониженным напряжением, а не с напряжением программирования V_{pp} , равным 11—13 В. Для программирования микросхемы при 5-вольтовом напряжении должен сначала устанавливаться бит конфигурации **Low Voltage Program** (Низковольтное программирование) в состояние **Enable** (Разрешение). Если высокий уровень приложен к выводу PGM, тогда PIC-контроллер также может быть записан в режиме низковольтного программирования. Это может иметь смысл, например, если программный код должен обновляться в готовом устройстве пользователя. Пользователь, как правило, не располагает программатором, чтобы запрограммировать микроконтроллер. Чтобы при программировании исключить неприятности, нужно стараться использовать этот вывод только как выход, который идет на вход внешней микросхемы. Дополнительно можно подключать еще *согласующий резистор* (Pull-Down Resistor) примерно от 10 кОм, чтобы вывод, если это требуется при отсутствии сигнала, имел бы только высокий логический уровень. Если, например, выход внеш-

него триггера Шмитта подключают к этому выводу, то никогда нельзя быть уверенным, какой уровень приложен на момент программирования к этому выводу.

На вход $\overline{\text{MCLR}}/V_{PP}$ микроконтроллера подается высокое напряжение программирования. Если это напряжение более 9 В, то внутренним аппаратным средствам сообщается, что должна программироваться Flash-память. После программирования этот вывод используется как вывод сброса. Если во время работы на этот вывод подается низкий уровень, то происходит сброс и программа начинается сначала. Чтобы избежать непреднамеренного сброса, на этом выводе должен всегда присутствовать высокий логический уровень. Но будьте осторожны: поскольку вывод не может быть подключен непосредственно к рабочему напряжению V_{DD} , т. к. иначе высокое напряжение программирования может попасть на вывод источника питания и вывести его из строя. Поэтому вывод V_{PP} должен подключаться к выводу V_{DD} (питанием) через резистор. Значение его сопротивления, как правило, должно быть равно 10 кОм. Таким образом, управлять сбросом можно с помощью кнопки или переключки (Jumper), замыкающей вывод V_{PP} на "землю". Полную монтажную схему для надежного программирования можно взять на рис. 4.1.

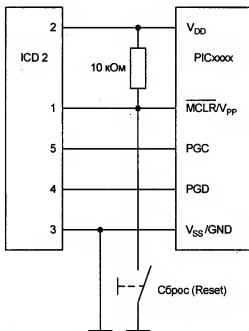


Рис. 4.1. Схема подключения отладчика-программатора ICD 2 к PIC-микроконтроллеру

4.2. Процесс программирования

Процесс программирования состоит из различных этапов. Чтобы записать новую программу во Flash-память микроконтроллера, память должна быть перед этим очищена. Кроме того, после записи хотелось бы быть уверенным в том, что данные действительно сохранены в памяти программ. Поэтому записанные данные считываются еще раз и сравниваются с данными на персональном компьютере.

Само собой разумеется, можно запускать каждый шаг в отдельности и таким образом записывать данные в микроконтроллер, а затем проверять их. Тем не менее, как правило, программирование протекает автоматически, и необходимые шаги запускаются по очереди. Для возможности передачи данных из персонального компьютера в PIC-контроллер сначала нужно инициализировать внутрисхемный отладчик MPLAB ICD 2 и выполнить соответствующие соединения. После подключения средой проектирования MPLAB проверяется правильность присоединения микроконтроллера. Если при настройке параметров был выбран ошибочный тип микроконтроллера, например, PIC16F876 вместо PIC16F876A, то в таком случае выводится предупреждение. В большинстве случаев это не критично и PIC-контроллер все же может программироваться. Однако в единичных случаях при выполнении программирования могут появиться проблемы. В среде проектирования MPLAB все необходимые команды для программирования микроконтроллера имеются в меню **Programmer** (Программатор). При выборе этих команд может выполняться ряд определенных действий.

- ☐ **Erase Part.** По этой команде очищается внутренняя Flash-память микроконтроллера. В памяти после процесса очистки будут находиться только единицы.
- ☐ **Blank Check.** Чтобы проверить, успешно ли прошла очистка памяти программ, применяется эта команда, с помощью которой выполняется проверка пустых, незаписанных мест в памяти. При этом содержимое всей Flash-памяти читается и проверяется, находится ли в каждой ячейке памяти 1. Если нет, т. е. не все ячейки очищены, то в памяти программ микроконтроллера еще находится старая программа. Такое может случиться, например, при передаче с помехами, когда используется слишком длинный кабель между программатором и схемой. Если такое происходит, то нужно проверить подключение и повторить процесс очистки памяти.
- ☐ **Program.** После того как в памяти программ больше не имеется данных, с помощью этой команды можно записывать программу во Flash-память. Программа, естественно, должна быть заранее загружена и скомпилирована. При программировании PIC-контроллера биты конфигурации про-

граммируются только в самом конце. После окончания программирования автоматически запускается перепроверка программного кода и битов конфигурации.

- ❑ **Read.** По этой команде, если чип не защищен от чтения с помощью бита конфигурации **Code Protect** (Код защиты), читается общее содержимое памяти. В случае очищенного микроконтроллера из каждой ячейки памяти программ читается значение 0x3FFF. Если программа находится в памяти, то виден бинарный кодируемый тип кода и дизассемблированные команды ассемблера, естественно без комментария и идентификаторов переменных, которые указывались в исходном коде. Чтобы увидеть программный код, нужно с помощью меню **View** (Обзор) и пункта **Program Memory** (Память программ) открыть одноименное окно **Program Memory** (Память программ).
- ❑ **Verify.** Чтобы не сравнивать каждую строку в отдельности с запрограммированным кодом, выберите эту команду и среда проектирования MPLAB автоматически проверит и сравнит данные.

После того как память программ будет успешно записана, можно снять рабочее напряжение и подать его снова без потери данных.

4.3. Биты конфигурации

К регистру, в котором хранятся биты конфигурации, можно обращаться только во время программирования. После того как чип запрограммирован, эти биты больше не могут изменяться. Чтобы сделать изменение в этих битах, весь чип должен быть сначала очищен, а затем снова запрограммирован с измененными настройками. Это действительно имеет смысл, поскольку в противном случае имела бы возможность, просто сбросив бит защиты от чтения, легко прочесть из памяти, ранее защищенные данные.

Биты конфигурации могут быть установлены либо непосредственно в программном коде, либо с помощью пункта **Configuration Bits** (Биты конфигурации) меню **Configure** (Конфигурация). Соответственно рекомендации нужно устанавливать биты в программном коде. Таким образом, к более позднему моменту программирования не забывают активировать защиту от чтения. Чтобы устанавливать биты конфигурации в программном коде, можно использовать настройки по умолчанию из дополнительного так называемого заголовочного файла с расширением `inc`, расположенные в конце файла. Далее в разд. 4.3.9 приведен пример возможных настроек для битов конфигурации, взятых из Include-файла. Распознать их можно по символу нижнего подчеркивания перед наименованием битов.

Регистр с установленными битами конфигурации имеет 14 битов. Чтобы установить биты конфигурации в программном коде, используют директиву ассемблера `__CONFIG`. После нее перечисляются константы для настройки битов конфигурации следующих друг за другом через символ "&".

Пример: `__CONFIG _CP_OFF & _WDT_OFF & _BODEN_OFF & ...`

4.3.1. Генератор

Микроконтроллер не может работать без тактовых импульсов, т. к. для обеспечения различных возможностей арифметического устройства необходима синхронизация. Выбор типа генератора зависит от требований к временной точности тактовых импульсов. Если требуется всего лишь подключать различные выходы в зависимости от входных сигналов, то часто достаточно простого RC-генератора (RC — резистивно-емкостной генератор). Если на основе микроконтроллера выполнен секундомер для измерения скорости, то требования к временной разрешающей способности значительно выше и поэтому требуется использовать кварцевый резонатор.

Таким образом, микроконтроллер PIC16F876A может работать в одном из четырех режимов тактового генератора.

RC: Resistor/Capacitor (`_RC_OSC`)
LP: Low-Power Crystal (`_LP_OSC`)
XT: Crystal/Resonator (`_XT_OSC`)
HS: High-Speed Crystal/Resonator (`_HS_OSC`)

Режим тактового генератора указывается в битах конфигурации. Если микроконтроллер эксплуатируется с кварцевым резонатором, то для тактовых частот менее 4 МГц используют режим XT. Как правило, этой тактовой частоты вполне достаточно для множества схем, и применяемые кварцы обеспечивают необходимую точность. Для более скоростных приложений, которые нуждаются в большей вычислительной производительности, устанавливают высокочастотный режим (режим HS) для диапазона от 4 до 20 МГц. Для применения кварца необходимы еще два дополнительных конденсатора, подключенных к выводам OSC1 и OSC2 микроконтроллера. При кварце на 4 МГц значения конденсаторов должны быть между 15 и 68 пФ.

Режим LP используется для кварцев с незначительной частотой, а именно менее 500 кГц. При использовании низкой тактовой частоты потребление тока от источника микроконтроллером также незначительно. Часто микроконтроллер переходит в режим ожидания и периодически выходит из него, чтобы проверять, имеются ли новые данные в наличии. В режиме ожидания (англ. SLEEP) или иначе в режиме сверхнизкого энергопотребления процессор не нуждается в полной вычислительной производительности и поэтому

может работать на низкой тактовой частоте. Как правило, для низкочастотного режима (режим LP — Low-Power) используется так называемый *часовой кварц* с частотой 32,768 кГц. Значение частоты на первый взгляд выглядит необычно. Но если это значение разделить 15 раз на 2, то получится такт, равный точно одной секунде. Поэтому кварцевый резонатор так называется — *часовой кварц*.

При создании оптимизированной по стоимости схемы, которая не предъявляет высокие требования к точности генератора, может использоваться внешний RC-генератор. При этом, к сожалению, точно рассчитать частоту генератора нельзя, поскольку она изменяется в зависимости от рабочего напряжения, температуры и применяемых комплектующих изделий. Так при напряжении питания 5 В, сопротивлении резистора 5 кОм и емкости конденсатора 100 пФ получается тактовая частота, примерно равная 1,3 МГц. Кроме того, существует ряд PIC-контроллеров, которые имеют свой внутренний RC-генератор. Он скорректирован производителем и может использоваться без внешних комплектующих изделий. При этих микроконтроллерах имеется специальный регистр, в котором содержится некоторая постоянная величина. С помощью этого регистра тактовая частота также может настраиваться в определенных пределах. Однако надо обращать внимание на то, что перед очисткой микроконтроллера этот регистр обязательно был бы сохранен и после программирования снова записан обратно в микроконтроллер. Это необходимо для того, чтобы можно было работать, используя скорректированный генератор.

Следует заметить, что схема с использованием микроконтроллера не обязательно должна иметь собственный генератор. Тактовые импульсы могут подаваться и от внешнего генератора. В этом случае внешние тактовые сигналы подаются на вывод OSC1, а вывод OSC2 остается неподключенным.

4.3.2. Сторожевой таймер

Сторожевой таймер WDT от англ. Watchdog-Timer, где Watchdog дословно переводится как "сторожевая собака", ведет себя аналогичным образом, т. е. как сторож. Он выполняет сторожевые функции в микроконтроллере. Если этот таймер активирован, то в фоновом режиме осуществляется отсчет времени. Когда отсчет времени завершится, т. е. произойдет переполнение таймера, процессор сбрасывается и программа начинается сначала. При работе в нормальном режиме требуется не допускать этого, и поэтому надо периодически сбрасывать таймер и позволять ему стартовать сначала. Если же процессор попадает в бесконечный цикл, микроконтроллер автоматически сбрасывается без помощи пользователя. Чтобы активировать сторожевой таймер, устанавливают соответствующий бит конфигурации `_wdt_on`, а при `_wdt_off` — таймер деактивируют.

4.3.3. Таймер включения питания

С помощью таймера включения питания PWRT (англ. Power Up Timer) микроконтроллер после подачи напряжения питания удерживается в состоянии сброса в течение 72 мс. Следовательно, процессор не начинает работу сразу же с выполнения программы, а ждет еще определенное короткое время. Это имеет смысл, когда на чип микроконтроллера подается рабочее напряжение питания. Поскольку в цепи напряжения питания устройства часто устанавливаются конденсаторы большой емкости, которые в течение некоторого определенного времени должны полностью зарядиться, прежде чем рабочее напряжение не достигнет номинального значения. Чтобы быть уверенным, что на процессор подано номинальное рабочее напряжение, включают таймер включения питания PWRT. Кроме того, вследствие этого обеспечивается подача верного питающего напряжения на аналого-цифровой преобразователь (АЦП). Если возрастание питающего напряжения до номинального рабочего напряжения продолжается около 1 мс, то при тактовой частоте 4 МГц уже выполнились бы примерно 1000 команд. Поэтому при проблемах в начале программы нужно сначала проверить, существуют ли стабильные состояния в микроконтроллере и не начинает ли он свою работу раньше времени, указанного в спецификации. Если требуется активировать таймер включения питания, то в программном коде устанавливают константу `_PWRT_ON` и отключают таймер при значении `_PWRT_OFF`.

4.3.4. Обнаружение провала напряжения

Во время нормального режима работы к микроконтроллеру всегда должно быть приложено достаточное значение постоянного напряжения. Если же питающее напряжение по влиянию каких-нибудь внешних воздействий падает ниже определенного значения, то эксплуатация процессора будет в не стабильном диапазоне и могут появляться ошибки при выполнении команд. Это событие может вызываться, например, кратковременным коротким замыканием во внешнем управляющем интерфейсе. Чтобы определить такие события, используют так называемый механизм обнаружения провала напряжения (англ. Brown-Out Detect). Под *провалом напряжения* понимают состояние, при котором напряжение питания становится ниже минимального предельного значения, гарантирующего нормальную работу устройства. Таким образом, если рабочее напряжение падает на заранее установленное время (примерно 100 мкс) ниже порога (примерно 4 В), вызывается сброс и программа начинается сначала после времени ожидания примерно 72 мс. Эта возможность активируется при помощи константы `_BODEN_ON`, разрешающего механизм обнаружения провала напряжения (англ. Brown-Out Detect Enable), и деактивируется при `_BODEN_OFF`.

4.3.5. Низковольтное программирование

С помощью бита конфигурации LVP (англ. Low Voltage Program), разрешающего режим низковольтного программирования микроконтроллера, пользователю предоставляется возможность выполнять программирование с использованием низкого напряжения, соответствующего стандартному диапазону для напряжения питания микроконтроллера. Если бит конфигурации активирован `_LVP_ON`, то вывод RB3 / PGM используется в качестве вывода PGM для режима низковольтного программирования. Это значит, что если на вывод PGM подается высокий логический уровень, то микроконтроллер может программироваться. Если бит конфигурации деактивирован, константа имеет значение `_LVP_OFF`, то вывод RB3 ведет себя как цифровой вывод порта ввода/вывода и для программирования Flash-памяти на вывод `MCLR/VPP` должно быть подано относительно высокое напряжение, приблизительно 12 В.

4.3.6. Защита чтения данных из EEPROM-памяти

Для запрета чтения данных, находящихся во внутренней EEPROM-памяти, сбрасывают бит CPD (англ. Code Protection Data) в регистре конфигурации (`CPD = 0`). Это можно выполнить программно в исходном коде, задав для бита конфигурации CPD значение ON, т. е. введя константу `_CPD_ON`. Если код защиты активирован, то данные больше не могут быть прочитаны и находятся в распоряжении только лишь внутренней программы. Таким образом можно защищаться против некомпетентного чтения аналогично установке пароля. Если же защита от чтения не требуется, то допускают свободное чтение указанием значения `_CPD_OFF`.

4.3.7. Запись Flash-памяти программы

Flash-память в PIC-контроллере может использоваться также для сохранения данных во время выполнения программы. С одной стороны это элегантное решение может лучше использовать собственную память. А с другой стороны эта возможность порождает опасность того, что в программный код может записаться ошибка, которая сделает программу более невыполнимой. В этом случае пришлось бы снова выполнять программирование микроконтроллера. Чтобы предотвратить запись поверх программного кода, разбиение Flash-памяти имеет 4 варианта областей, которые защищаются битами конфигурации от непреднамеренной записи. Защищаемый диапазон устанавливается с помощью двух битов конфигурации WRT0 и WRT1 (англ. Write).

Для микроконтроллера PIC16F876A имеются следующие варианты разбиения Flash-памяти:

- ☐ `_WRT_OFF`. При `WRT1 = 1` и `WRT0 = 1` защита от записи выключена, и запись может быть во всю Flash-память программ (`WRT1 = 1` и `WRT0 = 1`).
- ☐ `_WRT_256`. При `WRT1 = 1` и `WRT0 = 0` область памяти с адресами между `0x0000` и `0x00FF` защищена от записи. Память в диапазоне адресов между `0x0100` и `0x1FFF` может использоваться как память данных.
- ☐ `_WRT_1FOURTH`. При `WRT1 = 0` и `WRT0 = 1` первая четверть памяти (`0x0000` до `0x07FF`, т. е. страница 0) защищена от записи. Область памяти в диапазоне адресов между `0x0800` и `0x1FFF` (страница 1, 2 и 3) может быть записана посредством управляющего регистра `EECON`.
- ☐ `_WRT_HALF`. Если программа использует только половину памяти с младшими адресами (от `0x0000` до `0x0FFF`, т. е. страница 0 и 1), то она может быть защищена с помощью битов `WRT1 = 0` и `WRT0 = 0`. В этом случае другая половина памяти со старшими адресами (от `0x1000` до `0x1FFF`, т. е. страница 2 и 3) может использоваться для хранения данных.

4.3.8. Защита кода

После разработки какого-либо устройства, как правило, хотят защитить его программное обеспечение от некомпетентного чтения. Для этого в регистре конфигурации сбрасывают бит 13, т. е. бит защиты кода `CP` (англ. Code Protection). Теперь сохраненный в чипе программный код больше не может читаться и это препятствует созданию копии программы. При установке этого бита защита кода будет отключена. Если производятся изменения в коде, оригинальный код должен редактироваться и загружаться снова в память. В исходном коде защита чтения активируется путем задания константы `_CP_ALL` и деактивируется при `_CP_OFF`.

4.3.9. Обзор битов конфигурации

Если никакие биты конфигурации не были запрограммированы, т. е. не были указаны в исходном коде или были забыты, то при чтении будут читаться как логические 1 и вследствие этого все защиты в регистре конфигурации будут отключены. Поэтому при необходимости защита от записи и чтения должна включаться дополнительно. Далее приведен список констант, указываемых в программном коде для установки битов конфигурации микроконтроллера PIC16F876A с помощью директивы `CONFIG`, с описанием их назначения.

- ☐ `_RC_OSC / _HS_OSC / _XT_OSC / _LP_OSC` — выбор типа генератора;
- ☐ `_WDT_ON / _WDT_OFF` — активация/деактивация сторожевого таймера;

- ☐ `_PWRTE_ON` / `_PWRTE_OFF` — активация/деактивация таймера включения питания `PWRT`;
- ☐ `_BODEN_ON` / `_BODEN_OFF` — активация/деактивация сброса по снижению (по провалу) напряжения питания;
- ☐ `_LVP_ON` / `_LVP_OFF` — активация/деактивация режима низковольтного программирования;
- ☐ `_CPD_ON` / `_CPD_OFF` — активация/деактивация защиты чтения данных из EEPROM-памяти;
- ☐ `_WRT_OFF` / `_WRT_256` / `_WRT_1FOURTH` / `_WRT_HALF` — варианты выбора защищаемой области Flash-памяти;
- ☐ `_CP_ALL` / `_CP_OFF` — активация/деактивация защиты чтения программного кода.

Описанные константы относятся к микроконтроллеру PIC16F876A. В других типах микроконтроллеров эти константы или имеются частично, или вовсе отсутствуют. Для определения констант они могут быть взяты из дополнительного Include-файла с расширением `inc` для конкретного микроконтроллера.

4.4. Микроконтроллеры OTP-типа

Ранее при описании программирования предполагалось, что программный код сохраняется во внутренней Flash-памяти микроконтроллера. Однако эта память по сравнению с памятью, которая может программироваться только однократно, относительно дорогая. Поэтому многие микроконтроллеры имеют память, которая может быть записана только один раз. Тогда при использовании другой программы нужно удалять микросхему и устанавливать новую с актуальной программой. Поэтому тип микроконтроллера OTP (One Time Programmable) означает однократно программируемый. Микрочипы с типом OTP обозначаются буквой "C" (например, PIC16C622A).

5. Монтажная плата

Микроконтроллер всегда тесно связан с аппаратными средствами, т. к. программное обеспечение программируется специально для конкретной схемы. Если программное обеспечение создается для персонального компьютера, то не обязательно нужно знать, какая видеокарта или процессор используются. Другое дело при программировании микроконтроллера. Здесь очень важно знать, какой индикатор (дисплей) подключается и к какому выводу подсоединены те или иные кнопки и светодиоды. Чтобы программное обеспечение можно было испытывать на настоящих аппаратных средствах, далее представляется схема монтажной платы, с помощью которой могут испытываться и расширяться все примеры. Чтобы распечатать электрическую схему соединений для работы, найдите на компакт-диске соответствующий документ в формате PDF. Кроме того, там же имеется электрическая схема и чертеж размещения деталей в формате программы Eagle. Электрическая схема и чертеж размещения элементов создавались с помощью демонстрационной версии программы Eagle, которую для частных целей можно бесплатно загрузить с веб-страницы производителя (www.cadsoft.de). Электрическая схема и соответственно чертеж размещения элементов может изменяться и таким образом расширяться в зависимости от потребностей пользователя. Для уменьшения трудоемкости при создании собственной монтажной платы можно заказать печатную плату на сайте www.edmh.de.

5.1. Описание схемы аппаратных средств

Монтажная плата состоит из различных частей схемы, которые связаны с микроконтроллером PIC16F876A. Плата сконструирована так, что имеется возможность демонстрации многих свойств микроконтроллера. Обмен данными между монтажной платой и персональным компьютером осуществля-

ется по последовательному интерфейсу. Могут записываться данные во внешнюю EEPROM-память по интерфейсу I²C, измеряться аналоговые напряжения, анализироваться инфракрасные сигналы и выводиться сообщения на жидкокристаллический индикатор. На плате предусмотрены также 4 кнопки и 4 светодиода, с помощью которых можно соответственно влиять на состояние микроконтроллера или обеспечивать индикацию его состояний.

5.1.1. Блок питания

Для питания микроконтроллера PIC16F876A нужен источник с напряжением питания 5 В. Чтобы оно было всегда неизменно, на монтажной плате имеется стабилизатор напряжения LM7805 (рис. 5.1). Монтажная плата может питаться от нерегулируемого источника питания. Чтобы на стабилизаторе напряжения не росла рассеиваемая мощность, нужно питать плату напряжением от 7 до 12 В. Оба керамических конденсатора C10 и C11 подавляют высокочастотные помехи, а электролитический конденсатор C12 емкостью 47 мкФ сглаживает возможные пульсации выходного напряжения.

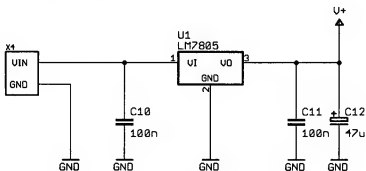


Рис. 5.1. Блок питания

5.1.2. Интерфейс программирования

Для осуществления более универсального программирования микроконтроллера он программируется посредством 5-контактного разъема X2 (рис. 5.2). С помощью подтягивающего резистора (R9) на выводе MCLR/V_{PP} постоянно поддерживается высокий логический уровень. Если же программа, при выполнении тестов попадает в бесконечный цикл, то, используя перемычку J3, можно вызывать сброс микроконтроллера.

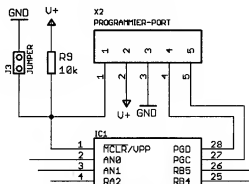


Рис. 5.2. Программный интерфейс

5.1.3. Генерация тактовых импульсов

Тактовые импульсы микроконтроллера формируются с помощью кварцевого резонатора с частотой 4 МГц (рис. 5.3). Для генерации тактовых импульсов помимо кварцевого резонатора требуются два керамических конденсатора (C8 и C9).

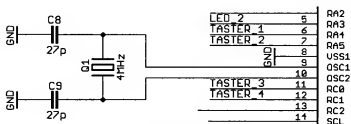


Рис. 5.3. Генерация такта

При монтаже схемы важно, чтобы конденсаторы подключались к земле по возможности более коротким путем. Чтобы это сделать возможным, PIC-контроллер располагает дополнительным выводом с землей GND (вывод 8).

5.1.4. Задание аналоговых напряжений

С помощью двух потенциометров R1 и R2 (рис. 5.4) имеется возможность задавать аналоговые входные напряжения, которые можно изменять от 0 В до 5 В. Если же надо использовать внешние аналоговые напряжения, то нужно отключить потенциометры от аналоговых входов AN0 и AN1, вынув перемычки J1 и J2. Если присоединяется внешний источник напряжения, то надо обращать внимание на то, чтобы его сопротивление было меньше 2,5 кОм,

поскольку иначе время преобразования будет увеличено и больше не будет совпадать со значением, указанным в спецификации.

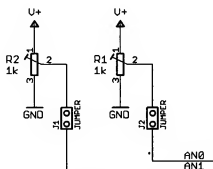


Рис. 5.4. Аналоговые напряжения

5.1.5. Кнопки

При помощи кнопок S1—S4 (рис. 5.5) на входы микроконтроллера можно подавать четыре статических сигнала, низкого логического уровня при нажатии и высокого логического уровня при отпускании кнопок. Если же ни одна

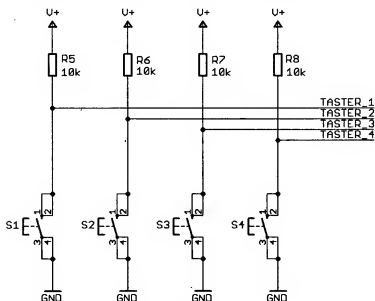


Рис. 5.5. Кнопки

из кнопок не нажата, то на всех выходах блока кнопок будут присутствовать сигналы высокого логического уровня благодаря подтягивающим резисторам R5—R8, подключенным к напряжению питания.

5.1.6. Индикация выходных сигналов на светодиодах

Светодиоды LED1—LED4 (рис. 5.6) индикации управляются выходными сигналами ключей, выполненных на транзисторах T1—T4. Принципиально светодиоды могли бы присоединяться непосредственно к двум выходам порта А и В. Однако транзисторы позволяют коммутировать более высокие токи, и поэтому имеется возможность простого расширения приведенной схемы. Токи светодиодов при открытых соответствующих транзисторах с помощью добавочных резисторов R11, R13, R15 и R17 ограничиваются примерно на уровне 10 мА.

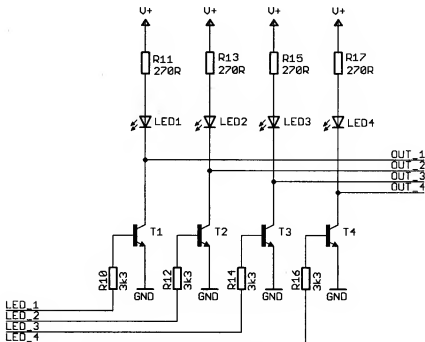


Рис. 5.6. Схема управления светодиодной индикации

5.1.7. Приемник инфракрасного излучения

На монтажной плате также установлен приемник инфракрасного (ИК) излучения. Вследствие этого микроконтроллер может управляться при помощи *инфракрасного дистанционного управления (телеуправления)*. Для реализации разнообразных схем телеуправления можно использовать различные приемники. В этой книге в приведенных примерах используется протокол RC5. Этот протокол очень распространен в Европе и часто применяется во многих схемах. Данное дистанционное управление работает на несущей частоте 36 кГц, поэтому применяется приемник инфракрасного излучения TSOP1736 производства компании Vishay (рис. 5.7). Если телеуправление работает на другой несущей частоте, то здесь может применяться другой приемник. Нужно обращать внимание лишь на то, чтобы выходной сигнал имел 5-вольтовый уровень.

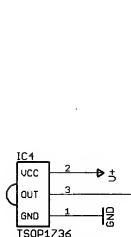


Рис. 5.7. Приемник инфракрасного излучения

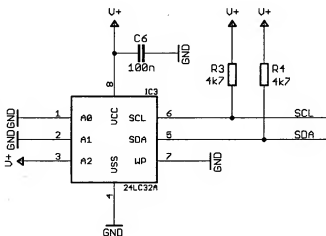


Рис. 5.8. EEPROM-память

5.1.8. EEPROM-память

Для сохранения больших объемов данных применяется внешняя EEPROM-память (рис. 5.8). В ней могут сохраняться данные посредством двухпроводного последовательного интерфейса I²C, поддерживаемого микросхемой памяти. Керамический конденсатор C6 уменьшает высокочастотные помехи в цепи питания и должен размещаться по возможности как можно ближе к выводу V_{CC} для подключения напряжения питания. Резисторы R3 и R4 нужны для формирования высокого логического уровня на линиях шины I²C при

условии, что присоединенные к ней элементы не активны. Если одно из устройств переключает на линии сигнал на низкий логический уровень, то все другие элементы, подключенные к шине I²C, определяют, что по интерфейсу происходит обмен сообщениями и не посылают данные.

5.1.9. Интерфейс RS-232

Для связи с персональным компьютером посредством последовательного интерфейса RS-232 требуется еще один дополнительный элемент, поскольку интерфейс RS-232 работает с уровнями сигналов от ± 8 до ± 12 В. Чтобы эти уровни смогли генерироваться из стандартных ТТЛ-сигналов от 0 до 5 В, используют приемопередатчики-драйверы последовательного интерфейса. В данном случае применяется микросхема MAX232A (рис. 5.9). Внутри этого элемента преобразование уровней сигналов осуществляется при помощи конденсаторов C2—C5. Устранение помех в цепи напряжения питания происходит с помощью конденсатора C1. Выводы 13 и 14 связаны с гнездами 9-контактного разъема Sub-D. С помощью выводов 11 и 12 приемопередатчики-драйверы микросхемы MAX232A связаны с микроконтроллером.

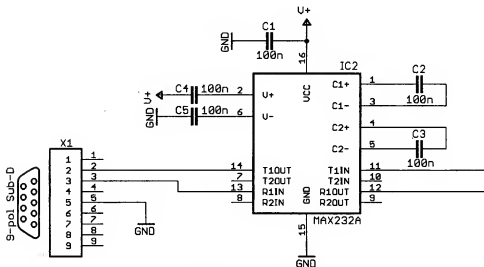


Рис. 5.9. Интерфейс RS-232

5.1.10. Жидкокристаллический индикатор

Для удобной индикации сообщений, результатов измерений или прочих текстов пользователю наверняка потребуется специальный жидкокристал-

личный индикатор (дисплей). На данной монтажной плате используется индикатор, отображающий 2 строки по 16 символов в каждой. Жидкокристаллический индикатор связан с микроконтроллером только по 4 линиям. Используемый индикатор, выпускаемый компанией Electronic Assembly (EA DOGM162), имеет преимущество в том, что с его помощью можно гибко выбирать между разными оттенками отображения, задавая по выбору определенную фоновую подсветку. Ток для фоновой подсветки постоянно установлен с помощью сопротивления резистора R18. Конденсатор C7 предназначен для устранения помех в цепи питания индикатора.

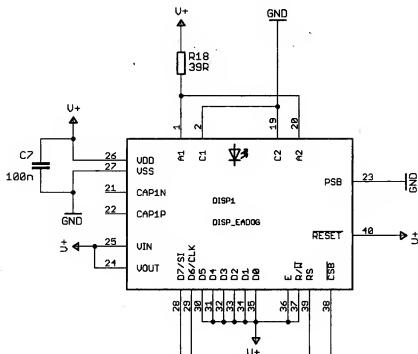


Рис. 3.10. Жидкокристаллический индикатор

5.1.11. Разъем для расширения

Расширение монтажной платы возможно с помощью предусмотренного разъема X3 (рис. 5.11). Например, к этому разъему расширения посредством плоского ленточного кабеля может подключаться такая же вторая монтажная плата для разработки. На разъеме расширения имеются контакты 4 цифровых входов, 4 выходов, 2 аналоговых входов, а также контакты шины I²C. При необходимости на второй плате может использоваться дополнительный ин-

дикатор. Если же индикатор не нужен, то в распоряжении будет еще 4 входа или выхода.

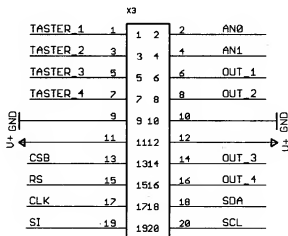


Рис. 5.11. Разъем для расширения монтажной платы

5.2. Программное обеспечение

С помощью программного обеспечения указывается, какой вывод микроконтроллера должен функционировать как вход, а какой как выход. Также должно определяться, с помощью каких выводов должны измеряться аналоговые напряжения и при помощи какого генератора должен работать микроконтроллер. Поскольку все примеры относятся к ранее описанным аппаратным средствам, то в этом разделе книги достаточно подробно описывается инициализация микроконтроллера. В этом случае определения и константы используются во всех примерах без следующего объяснения. Следует заметить, что в каждом примере на приложенном компакт-диске эти определения расположены в самом начале программного кода.

5.2.1. Подключение внешних файлов

Первая директива `list` изменяет ряд параметров при компиляции исходного файла и генерации файла листинга. В частности, параметр `p` в директиве `list` ассемблера устанавливает, какой микроконтроллер должен использоваться в данном программном коде.

```
list p=16f876a
```

Поскольку каждый микроконтроллер имеет свои функциональные особенности, то в программе на ассемблере используются predetermined константы, которые могут быть включены в файл с расширением `inc`. В этом файле устанавливается, например, что `PORTA` находится по адресу `0x05`, а флаг нулевого результата `Z` расположен в регистре состояния `STATUS` на месте бита 2. С помощью директивы `#include <...>` можно указывать ассемблеру дополнительные файлы для использования, которые читаются аналогично исходному тексту программы в то место программного кода, где расположена директива. Таким образом, имеется возможность в собственном файле делать макроопределения и включать их для каждого нового проекта в программу.

```
#include <p16f876a.inc>
```

5.2.2. Биты конфигурации

С помощью битов конфигурации микроконтроллеру сообщается, какие области памяти должны быть защищены, какой использовать генератор и как должен себя вести микроконтроллер при сбросе. Биты конфигурации могут устанавливаться через специальные инструментальные программные средства или непосредственно в программном коде с помощью директивы `__CONFIG`. Посредством этой директивы ассемблеру сообщают, что далее следует список констант для установки битов конфигурации в указанные значения. Эти константы должны стоять в строке и следовать друг за другом через символ `&`.

```
__CONFIG _CP_OFF & _WDT_OFF & _BODEN_OFF & _PWRTE_OFF & _XT_OSC
& _WRT_OFF & _LVP_OFF & _DEBUG_ON & _CPD_OFF
```

5.2.3. Определения

С помощью директивы `#define` можно определить замену текста в листинге. Стоящий справа за директивой `#define` условный текст заменяется на следующую далее и реально поддерживаемую ассемблером последовательность символов, которая является подлинным текстом. Замена условного текста на подлинный осуществляется перед процессом компиляции. Таким образом, программный код на ассемблере может оформляться наиболее понятным образом.

```
=====
; Определения
=====
#define Taster_1    PORTA, 4      ;Кнопка S1 подключена к выводу RA4
#define Taster_2    PORTA, 5      ;Кнопка S2 подключена к выводу RA5
```



```

#define Taster_1    PORTC, 0    ;Кнопка S3 подключена к выводу RC0
#define Taster_4    PORTC, 1    ;Кнопка S4 подключена к выводу RC1
#define LED_1       PORTA, 2    ;Светодиод LED1 подключен к выводу RA2
#define LED_2       PORTA, 3    ;Светодиод LED2 подключен к выводу RA3
#define LED_3       PORTB, 4    ;Светодиод LED3 подключен к выводу RB4
#define LED_4       PORTB, 5    ;Светодиод LED4 подключен к выводу RB5
#define DISP_SI     PORTB, 0    ;Последовательные данные индикатора
#define DISP_CLK    PORTB, 1    ;Тактовые импульсы для индикатора
#define DISP_RS     PORTB, 2    ;Сигнал для индикатора, разделяющий
                                ; (H-уровень) и данные (B-уровень)
#define DISP_CSB    PORTB, 3    ;Сигнал "выбор кристалла"
                                ; #CSB (Chip Select) для индикатора
#define IR_RCV      PORTC, 2    ;Приемник инфракрасного излучения
                                ; подключен к выводу RC2

```

5.2.4. Переменные

В программном коде в секции *Переменные* (VARIABLEN) указываются регистры, которые используются в программе. В начале строки стоит идентификатор переменной или имя самостоятельно определенного регистра. За директивой EQU указывается значение или адрес регистра. Регистры специального назначения с фиксированными адресами (например, PORTA) определяются таким же способом.

```

;=====
; Переменные
;=====
w_temp EQU 0x70
status_temp EQU 0x71

```

5.2.5. Макрокоманды

Другой вид замены текста — это макрокоманды (*макросы*), т. е. последовательность инструкций и директив, которые могут быть вставлены в код программы, используя специальный запрос макроса. Большое преимущество макросов состоит в том, что с их помощью можно передавать значения, которые заменяются перед процессом трансляции. В начале строки стоит метка или иначе имя макроса, с помощью которого вызывается макрокоманда. Директива `macro` обозначает начало макрокоманды. Затем указываются транслируемые данные, которые имеются в используемой макрокоманде. В следующих строках перечисляются команды и директивы ассемблера, которые заменят имя макроса, указанное в программном коде. Конец макрокоманды обозначается с помощью директивы `endm`.

```

;=====
; Макросы
;=====
_BANK_0 macro          ;Выбор регистра из Банка 0
    BCF STATUS,RP0
    BCF STATUS,RP1
endm

_BANK_1 macro          ;Выбор регистра из Банка 1
    BSF STATUS,RP0
    BCF STATUS,RP1
endm

_BANK_2 macro          ;Выбор регистра из Банка 2
    BCF STATUS,RP0
    BSF STATUS,RP1
endm

_BANK_3 macro          ;Выбор регистра из Банка 3
    BSF STATUS,RP0
    BSF STATUS,RP1
endm

```

5.2.6. Начало программы

После окончания всех описаний ассемблеру должен быть сообщен адрес, с которого следующая программа должна сохраняться в микроконтроллере. Поскольку после сброса микроконтроллер начинает выполнение программы с адреса 0x000, то это и указывается директивой `ORG`. Следовательно, первая команда стоит на этом месте.

```

ORG 0x000              ;Адрес для передачи управления после сброса
clrf PCLATH             ;Сбросить биты страницы
goto start              ;Перейти к началу программы

```

После прерывания программный счетчик переходит не по адресу начала программы, а по адресу 0x004. Поэтому код для программы обслуживания прерывания также должен начинаться с этого адреса.

```

ORG 0x004              ;Адрес прерывания
movwf w_temp           ;Сохранить содержимое регистра W
movf STATUS,w          ;Переместить содержимое регистра состояния в рег. W
movwf status_temp      ;Сохранить содержимое регистра состояния STATUS
;Отсюда может начинаться код для программ обработки прерывания
movf status_temp,w     ;Получить копию содержимого регистра STATUS
movwf STATUS           ;Переместить эти данные назад в регистр STATUS
swapf w_temp,f         ;
swapf w_temp,w         ;Сохранить данные в W-регистре
retfie                 ;Возврат из прерывания

```

Далее, как правило, следует метка перехода (в данном случае метка `start`), которая указывалась в команде `goto` за командой `ORG 0x000`. Отсюда собственно и начинается основная программа.

```
start                ;Начало программы
;Инициализация
_BANK_0              ;Включить при запуске Банк 0
clrf PORTA            ;Установить все биты PORTA в 0
clrf PORTB            ;Установить все биты PORTB в 0
clrf PORTC            ;Установить все биты PORTC в 0
```

5.2.7. Инициализация

В начале инициализации в этом примере все выводы порта ввода/вывода `PORTA` настраивают в качестве цифровых. Выводы этого порта могут использоваться и в качестве аналоговых входов или линий интерфейса I²C, но после соответствующей настройки во время выполнения программы. Затем некоторые выводы портов ввода/вывода `PORTA`, `PORTB` и `PORTC` настраивают как входы, а некоторые как выходы.

```
_BANK_1              ;Переключиться на Банк 1 для регистра TRIS и ADCON1
movlw b'00000110'    ;Все выводы PORTA сделать цифровыми выводами I/O
movwf ADCON1
movlw b'11110011'    ;Настроить биты <3:2> порта ввода/вывода PORTA
movwf TRISA           ;как выходы, остальные как входы
movlw b'11000000'    ;Настроить биты <7:6> порта ввода/вывода PORTB
movwf TRISB           ;как выходы, остальные как входы
movlw b'11111111'    ;Настроить все выводы PORTC как входы
movwf TRISC
_BANK_0              ;Переключиться обратно на Банк 0
```

Инициализация завершена и начинает основной цикл.

```
main                  ;Начало основного цикла
...
Здесь стоят команды ассемблера программы
...
goto main            ;Выполнить программу сначала
```

В конце программы на ассемблере нужно задать еще одну директиву `END`, которая указывает на окончание команд программы. Код программы будет транслироваться именно до этого места.

```
END                  ;Конец программы
```


6. Входы и выходы

У каждого микроконтроллера имеются входы и выходы, через которые он может обмениваться информацией с внешним миром. У некоторых микроконтроллеров есть более 100 свободно поддающихся настройке входов и выходов, которые часто называются *GPIO* (General Purpose Input Output — входы и выходы общего применения). Кроме того, существуют и небольшие микроконтроллеры, которые имеют не более 4 входов или выходов (например, PIC10F222). Как правило, посредством различных регистров можно настраивать, какую функцию должны иметь эти выводы. Они могут использоваться как аналоговый вход или как цифровой вход и выход. Также они могут быть определены для специальной функции, например, для коммуникации по последовательному интерфейсу или для применения в качестве линий шины I²C. Следует заметить, что функции выводов могут изменяться также во время выполнения программы, вследствие чего возможно, что вывод определенный сначала как вход, после появления какого-либо события будет переопределен и будет функционировать уже как выход.

6.1. Расположение выводов PIC16F876A

Микросхема микроконтроллера PIC16F876A всего имеет 28 выводов, из них 22 вывода GPIO. Они используются в качестве обычного цифрового входа или выхода, либо посредством мультиплексора переназначены для выполнения специальной функции. Поскольку к 22 выводам ввода/вывода (I/O) нельзя обратиться с помощью одного 8-битного регистра, то они разделяются на 3 порта: PORTA, PORTB и PORTC. Порт A содержит выводы RA0—RA5, порт B — выводы RB0—RB7, а посредством порта C обращаются к выводам RC0—RC7. Шесть остальных выводов микросхемы используются для подключения источника питания (V_{DD} , V_{SS}), сброса ($\#MCLR$) и тактирования (OSC1, OSC2).

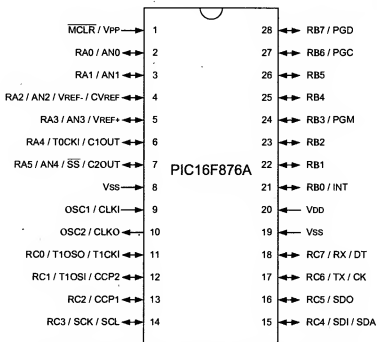


Рис. 6.1. Расположение выводов микроконтроллера PIC16F876A

ПРИМЕЧАНИЕ

Символ #, установленный слева от сокращенного обозначения выводов и слева от соответствующих сигналов, обозначает вывод или сигнал с активным низким логическим уровнем.

На рис. 6.1 по обозначениям выводов микроконтроллера PIC16F876A можно легко определить, сколько различных функций может выполнить каждый вывод. Поэтому требуется заранее определиться, какую именно функцию он должен реализовать. Это решение далее отразится при разработке электрической схемы и программного обеспечения. Как правило, это не вызывает проблемы, если вначале для вывода назначают определенную функцию и уже больше ее не изменяют. Однако может случиться так, что нужно обращаться к внешнему элементу посредством *шины* I²C (Inter Integrated Circuit Bus — последовательная шина данных для связи интегральных схем) и к другому элементу через *интерфейс SPI* (Serial Peripheral Interface — последовательный периферийный интерфейс). Поскольку эти функции для интерфейсов I²C и SPI определены совместно на выводах 14 и 15, то в этом случае для них во время выполнения программы требуется предусматривать переключение функций. Однако, чтобы чип, к которому обращаются посредством шины I²C, получал данные только этой шины, надо дополнительно предусмотреть

внешний переключатель, который передавал бы данные дальше к нужному элементу. Следовательно, вывод может быть использован для нескольких функций, но зачастую только с затратами на дополнительные аппаратные средства.

6.2. Обзор функций выводов

Для наиболее понятного и наглядного представления о каждом выводе на схеме расположения выводов все обозначения выводов приведены условными сокращениями. Чтобы лучше понять функцию, которая скрывается за этим сокращением, можно найти более детальное его пояснение в табл. 6.1. При проектировании, как правило, выводы порта А используются для аналоговых специальных функций, выводы порта В применяются только для цифровых входов и выходов, а вот выводы порта С используются для интерфейсов, предназначенных для связи с другими элементами и устройствами.

Таблица 6.1. Обзор функций выводов

Вывод	Обозначение	Направление	Назначение
1	#MCLR	I	Вход сброса микроконтроллера
	V _{PP}	P	Вход напряжения программирования
2	RA0	I/O	Вывод цифрового входа/выхода порта А (бит 0)
	AN0	I	Аналоговый вход (канал 0)
3	RA1	I/O	Вывод цифрового входа/выхода порта А (бит 1)
	AN1	I	Аналоговый вход (канал 1)
4	RA2	I/O	Вывод цифрового входа/выхода порта А (бит 2)
	AN2	I	Аналоговый вход (канал 2)
	V _{REF-}	I	Вход отрицательного опорного напряжения для АЦП
	CV _{REF}	O	Выход опорного напряжения компаратора
5	RA3	I/O	Вывод цифрового входа/выхода порта А (бит 3)
	AN3	I	Аналоговый вход (канал 3)
	V _{REF+}	I	Вход для положительного опорного напряжения для АЦП
6	RA4	I/O	Вывод цифрового входа/выхода порта А (бит 4); выход с открытым стоком (open-drain)
	T0CKI	I	Вход внешней синхронизации для таймера Timer0
	C1OUT	O	Выход компаратора 1

Таблица 6.1 (продолжение)

Вывод	Обозначение	Направление	Назначение
7	RA5	I/O	Вывод цифрового входа/выхода порта А (бит 5)
	AN4	I	Аналоговый вход (канал 4)
	#SS	I	Вход выбора микросхемы в режиме ведомого SPI
	C2OUT	O	Выход компаратора 2
8	V _{ss}	P	Общий вывод питающего напряжения, позволяет выполнить короткое соединение кварца с землей
9	OSC1	I	Вывод 1 для подключения кварцевого резонатора
	CLKI	I	Вход тактового сигнала от внешнего источника
10	OSC2	O	Вывод 2 для подключения кварцевого резонатора
	CLKO	O	В RC-режиме тактового генератора на выходе присутствует сигнал CLKO, равный 1/4 частоты на выводе OSC1
11	RC0	I/O	Вывод цифрового входа/выхода порта С (бит 0)
	T1OSO	O	Выход генератора таймера Timer1
	T1CKI	I	Вход внешнего тактового сигнала таймера Timer1
12	RC1	I/O	Вывод цифрового входа/выхода порта С (бит 1)
	T1OSI	I	Вход генератора таймера Timer1
	CCP2	I/O	Вход захвата, выход сравнения, выход ШИМ (PWM) модуля CCP2 (т. е. второго модуля захвата/сравнения/ШИМ)
13	RC2	I/O	Вывод цифрового входа/выхода порта С (бит 2)
	CCP1	I/O	Вход захвата, выход сравнения, выход ШИМ модуля CCP1 (Capture/Compare/PWM), т. е. первого модуля захвата/сравнения/ШИМ)
14	RC3	I/O	Вывод цифрового входа/выхода порта С (бит 3)
	SCK	I/O	Вход или выход для тактового сигнала в режиме SPI
	SCL	I/O	Вход или выход для тактового сигнала в режиме I ² C
15	RC4	I/O	Вывод цифрового входа/выхода порта С (бит 4)
	SDI	I	Вход для данных в режиме SPI
	SDA	I/O	Вход или выход для данных в режиме I ² C
16	RC5	I/O	Вывод цифрового входа/выхода порта С (бит 5)
	SDO	O	Выход для данных в режиме SPI

Таблица 6.1 (окончание)

Вывод	Обозначение	Направление	Назначение
17	RC6	I/O	Вывод цифрового входа/выхода порта C (бит 6)
	TX	O	Вывод передатчика асинхронного обмена USART
	CK	I/O	Вывод тактов синхронизации USART в синхронном режиме
18	RC7	I/O	Вывод цифрового входа/выхода порта C (бит 7)
	RX	I	Вывод приемника асинхронного обмена USART
	DT	I/O	Вывод данных USART в синхронном режиме
19	V _{ss}	P	Общий вывод питающего напряжения
20	V _{DD}	P	Положительное напряжение питания
21	RB0	I/O	Вывод цифрового входа/выхода порта B (бит 0)
	INT	I	Вход внешних прерываний
22	RB1	I/O	Вывод цифрового входа/выхода порта B (бит 1)
23	RB2	I/O	Вывод цифрового входа/выхода порта B (бит 2)
24	RB3	I/O	Вывод цифрового входа/выхода порта B (бит 3)
	PGM	I	Вход для разрешения режима низковольтного программирования
25	RB4	I/O	Вывод цифрового входа/выхода порта B (бит 4)
26	RB5	I/O	Вывод цифрового входа/выхода порта B (бит 5)
27	RB6	I/O	Вывод цифрового входа/выхода порта B (бит 6)
	PGC	I	Тактовый вход в режиме программирования
28	RB7	I/O	Вывод цифрового входа/выхода порта B (бит 7)
	PGD	I/O	Вход или выход данных в режиме программирования

Пояснение условных обозначений:

- ☐ I (англ. Input) — вход;
- ☐ O (англ. Output) — выход;
- ☐ I/O (англ. Input/Output) — вход или выход, в зависимости от установок регистра;
- ☐ P (англ. Power) — вывод источника электроэнергии;

6.3. Цифровые входы и выходы

Микроконтроллер оперирует только с цифровыми данными, т. е. нулями и единицами. Поэтому любой результат также представляется в цифровой форме. По этой причине практически любой вывод может быть использован как цифровой вход или выход. С помощью цифровых выходов можно переключать, например, светодиод (*LED* — Light Emitting Diode, светоизлучающий диод), который при этом будет показывать определенное состояние вывода. Выходной вывод микроконтроллера PIC16F876A в принципе может управлять светодиодом напрямую, поскольку цифровой выход без проблем может коммутировать ток до 10 мА, который, как правило, и потребляют обычные светодиоды. Если же необходим ток, превышающий 10 мА, то к выходу микроконтроллера нужно дополнительно подключить усилитель мощности, например, на транзисторе. Чтобы считывать цифровые сигналы, например, для контроля нажата кнопка или нет, выводы могут быть настроены на использование их в качестве цифровых входов. В этом случае выводы ведут себя как входы ТТЛ-микросхем.

Поскольку микроконтроллер после включения не может знать, какой его вывод должен функционировать в качестве входа или выхода, то это нужно ему указать заранее с помощью программного кода. После включения или после сброса контроллер исходит из того, что все его выводы будут определены как входы. Поэтому никакое напряжение на выводе также не выводится. Если этого не сделать, то могло бы быть короткое замыкание в том случае, если присоединенная микросхема на выходе генерирует высокий логический уровень, в то время как у PIC-контроллера на этом выводе сигнал низкого уровня. Та или иная соответствующая функция для выводов назначается, как правило, в стартовой последовательности при инициализации.

Будет вывод входом или выходом, указывается в регистрах управления передачей данных TRISA, TRISB и TRISC. Все эти 3 регистра находятся в банке 1. При инициализации нельзя забывать и про коммутацию банков памяти. Если вывод настраивают на вход, то в соответствующий разряд регистра записывается логическая 1. В противном случае, если вывод настраивают на выход, то в этот же разряд заносят логический 0. Для легкого запоминания можно применить следующую "шпаргалку": для настройки вывода на вход (Input) в соответствующий бит регистра управления надо записать 1, которая выглядит как I (Input), а для настройки вывода на выход (Output) надо записать 0, который выглядит как O (Output). Если, например, вывод RB2 должен быть входом, то в бит 2 регистра TRISB записывают 1, а чтобы настроить вывод RA4 на цифровой выход, в бит 4 регистра TRISA заносят 0.

Пример:

```
movlw b'11110011'      ;Настроить выходы RA3 и RA2 на выход,  
                        ;а остальные на вход  
movwf TRISA  
movlw b'11000000'      ;Настроить выходы RB7 и RB6 на вход,  
                        ;а остальные на выход  
movwf TRISB  
movlw b'11111111'      ;Все выходы порта C настроить на вход  
movwf TRISC
```

Как видно из примера, для настройки направления для выводов всех портов в принципе было бы достаточно всего шести команд. Однако некоторые выводы не только используются как цифровые входы или выходы, а могут применяться в качестве аналоговых входов. Поэтому помимо регистров, управляющих направлением передачи для выводов портов, необходимы еще дополнительные управляющие регистры, которые нужны для выбора аналогового или цифрового режима, а для аналогового режима еще и для управления АЦП.

Таким образом, чтобы использовать цифровые входы порта А, должен применяться еще один дополнительный регистр ADCON1 (A/D Control register 1). Этот регистр имеет 13 возможных вариантов для настройки выводов порта в качестве цифрового входа или выхода и 14 вариантов для определения выводов в качестве входов для аналоговых сигналов. При настройке функции выводов имеют значение только младшие 4 бита этого регистра, а именно PCFG [3...0]. Старшие его два разряда требуются только, если должно осуществляться аналого-цифровое преобразование. Все возможные значения регистра можно взять из описания, представленного в табл. 6.2. Следует заметить, что выводы AN7, AN6 и AN5 в контроллере PIC16F876A отсутствуют и имеются только в одноканальных контроллерах в корпусах с большим количеством выводов (PIC16F874A, PIC16F877A). Кроме того, вывод RA4 в отличие от всех других выводов порта А мультиплексирован не с аналоговым входом АЦП (AN4—AN0); а с входом T0CK1, предназначенным для внешних тактирующих импульсов таймера Timer0.

Пояснение условных обозначений (в табл. 6.2):

- ☐ A (англ. Analog) — аналоговый вход;
- ☐ D (англ. Digital) — цифровой вход или выход;
- ☐ V_{REF+} (англ. Voltage Reference) — вывод для положительного опорного напряжения;
- ☐ V_{REF-} (англ. Voltage Reference) — вывод для отрицательного опорного напряжения.

Таблица 6.2. Описание выводов в регистре ADCON1

Биты регистра ADCON1	Выводы порта E			Выводы порта A				
	RE2	RE1	RE0	RA5	RA3	RA2	RA1	RA0
PCFG [3...0]	AN7	AN6	AN5	AN4	AN3	AN2	AN1	AN0
0000	A	A	A	A	A	A	A	A
0001	A	A	A	A	V _{REF+}	A	A	A
0010	D	D	D	A	A	A	A	A
0011	D	D	D	A	V _{REF+}	A	A	A
0100	D	D	D	D	A	D	A	A
0101	D	D	D	D	V _{REF+}	D	A	A
0110	D	D	D	D	D	D	D	D
0111	D	D	D	D	D	D	D	D
1000	A	A	A	A	V _{REF+}	V _{REF-}	A	A
1001	D	D	A	A	A	A	A	A
1010	D	D	A	A	V _{REF+}	A	A	A
1011	D	D	A	A	V _{REF+}	V _{REF-}	A	A
1100	D	D	D	A	V _{REF+}	V _{REF-}	A	A
1101	D	D	D	D	V _{REF+}	V _{REF-}	A	A
1110	D	D	D	D	D	D	D	A
1111	D	D	D	D	V _{REF+}	V _{REF-}	D	A

Чтобы использовать все выводы порта A как цифровые входы или выходы, в младших 4 битах регистра ADCON1 должно быть двоичное значение 0110 или 0111. Если же используются 1 аналоговый вход (AN0) и 5 цифровых входов, можно применять установку в 1110. К сожалению, нельзя задать 2 аналоговых входа и 4 цифровых входа или выхода. На этом месте нужно решаться, устанавливать ли 3 аналоговых входа (0100), или использовать 2 аналоговых входа с выводом для опорного напряжения (0101).

В следующем примере выводы RA0, RA2 и RA3 устанавливаются в качестве цифровых входов, а выводы RA1, RA4 и RA5 в качестве цифровых выходов.

Пример:

```
movlw b'00000110'      ;Все выводы порта A сделать цифровыми входами
                        ;или выходами
movwf ADCON1
```

```
movlw b'00001101'      ;Настроить, выходы RA0, RA2 и RA3 в качестве
                          ;входов, а выходы RA1, RA4 и RA5 — выходов
movwf TRISA
```

Выходы порта В могут использоваться только как цифровые входы или выходы. Лишь во время процесса программирования или отладки у некоторых этих выводов имеется еще и другое предназначение. Единственный вывод с особенной функцией — это вывод RB0/INT. С помощью этого входа может быть вызвано внешнее прерывание по фронту входного импульса сигнала. Должно ли прерывание вызываться при возрастающем переднем фронте импульса (от низкого к высокому уровню) или при спадающем заднем фронте импульса (от высокого к низкому уровню), может быть выбрано в регистре OPTION_REG. В этом же регистре можно указать, должны ли для выводов порта В использоваться внутренние подтягивающие резисторы. Запретить же или разрешить внешнее прерывание можно с помощью специального управляющего регистра ITCON. Чтобы установить, какие выходы порта В будут входами, а какие выходами, нужно воспользоваться регистром TRISB. Для того чтобы действовать наверняка и быть уверенным, что сделаны правильные настройки для выполнения программы, нужно воспользоваться полным описанием регистра OPTION_REG.

Пример:

```
movlw b'01000000'      ;Значение инициализации OPTION_REG
movwf OPTION_REG
movlw b'11001010'      ;Выходы RB7, RB6, RB3, RB1 настроить как входы
movwf TRISB             ;Выходы RB5, RB4, RB2, RB0 настроить как выходы
```

В приведенном примере выходы RB7, RB6, RB3 определяются в качестве входов, поскольку они должны использоваться по возможности только для программирования. Кроме того, включены внутренние подтягивающие резисторы на входах порта В. Вследствие этого все входы с подтягивающим резистором имеют высокий логический уровень, если к этому выводу не подключен никакой внешний элемент. Можно отключать применение подтягивающего резистора установкой бита 7 регистра OPTION_REG в состояние логической 1.

Для выводов порта С настройки несколько сложнее, поскольку соответствующие ему выходы могут выполнять больше различных функций. До тех пор, пока выходы используются только как цифровые входы или выходы, их можно без проблем настроить посредством регистра TRISC. Если же во время выполнения программы хотят осуществить связь по шине I²C, то направление данных больше не будет соответствовать установкам в регистре TRISC. Направлением выводов в этом случае управляют аппаратные периферийные модули, находящиеся в чипе и работающие в режиме I²C. Вследствие

этого периферийные модули отменяют действия битов регистра TRISC, принудительно настраивая выходы на вход или выход. Как правило, все выходы во время программы выполняют определенную постоянную функцию, поэтому в начале программы порт C можно соответствующим образом инициализировать. В следующем примере показан вариант инициализации порта C. Выводы RC0 и RC1 должны использоваться как входы, а вывод RC2 как выход. Посредством выводов RC3 и RC4 должна осуществляться связь по шине I²C. Выводы RC6 и RC7 по последовательному интерфейсу дают возможность связи с персональным компьютером. Вывод RC5 не используется в схеме и поэтому определяется как вход. В примере показано только то, как настраиваются выходы в качестве цифровых входов/выходов. Инициализация интерфейсов гораздо сложнее и рассмотрена далее в последующих главах. Поэтому все выходы кроме RC2 настраиваются в качестве входов, чтобы не посылать какие-либо сигналы на периферию непосредственно после начала программы.

Пример:

```
movlw b'11111011'      ;Только вывод RC2 порта C
movwf TRISC             ;настроить в качестве выхода
```

Тогда, когда требуется меньшее количество выводов для портов, чем имеется в микроконтроллере, неиспользуемые выходы нужно настроить в качестве входов и предусмотреть для них в схеме подтягивающий или согласующий резистор. В этом есть свой резон, поскольку в этом случае входной вывод имел бы определенный логический уровень. Когда же вывод оставляют открытым, то экономят внешнее сопротивление, однако, тогда уже нельзя быть уверенным в том, верное ли значение будет прочитано из порта. Это особенно важно, например, если в программе выполняют сравнение некоторой 8-битной постоянной с полученным из порта значением, поскольку результат тогда может быть совершенно непредсказуемым. Если же эта проверка важна для последующего выполнения программы, тогда заранее даже не представлять, как она себя далее поведет.

6.4. Пример программы "Управление светодиодами"

До сих пор были показаны только отдельные части программного кода. Начиная с приведенного далее примера, будет все несколько иначе. С одной стороны, этот пример кода может быть симитирован в среде разработки MPLAB, а с другой стороны, его можно опробовать на реальном аппаратном устройстве, собранном по электрической схеме, которая была рассмотрена в главе 5. В данном примере четыре светодиода LED1—LED4 управляются

в зависимости от нажатия четырех кнопок S1—S4. В примере показано, каким образом осуществляется циклический опрос кнопок и как реализуется управление соответствующими выходами, чтобы включались и выключались светодиоды. Этот пример программного кода, как и все остальные примеры, находится на прилагаемом к книге компакт-диске.

```
main                                ;Начало основного цикла
    btfs Taster_1                  ;Запрос, нажатия на кнопку 1
    goto taster_1_gedrückt         ;Включить светодиод, соответствующий кнопке 1
    btfs Taster_2                  ;Запрос, нажатия на кнопку 2
    goto taster_2_gedrückt         ;Включить светодиод, соответствующий кнопке 2
    btfs Taster_3                  ;Запрос, нажатия на кнопку 3
    goto taster_3_gedrückt         ;Включить светодиод, соответствующий кнопке 3
    btfs Taster_4                  ;Запрос, нажатия на кнопку 4
    goto taster_4_gedrückt         ;Включить светодиод, соответствующий кнопке 4
    goto main                      ;Если ни на какую кнопку не нажимали,
                                ;то перейти в начало основного цикла

taster_1_gedrückt                   ;Кнопку 1 нажимали
    bsf LED_1                      ;LED 1 включить
    bcf LED_2                      ;отключить LED 2
    bcf LED_3                      ;отключить LED 3
    bcf LED_4                      ;отключить LED 4
    goto main                      ;перейти назад в основной цикл

taster_2_gedrückt                   ;Кнопку 2 нажимали
    bcf LED_1                      ;отключить LED 1
    bsf LED_2                      ;LED 2 включить
    bcf LED_3                      ;отключить LED 3
    bcf LED_4                      ;отключить LED 4
    goto main                      ;Перейти назад в основной цикл

taster_3_gedrückt                   ;Кнопку 3 нажимали
    bsf LED_1                      ;LED 1 включить
    bcf LED_2                      ;отключить LED 2
    bsf LED_3                      ;LED 3 включить
    bcf LED_4                      ;отключить LED 4
    goto main                      ;перейти назад в основной цикл

taster_4_gedrückt                   ;Кнопку 4 нажимали
    bsf LED_1                      ;LED 1 включить
    bsf LED_2                      ;LED 2 включить
    bsf LED_3                      ;LED 3 включить
    bsf LED_4                      ;LED 4 включить
    goto main                      ;Перейти обратно в основной цикл
END                                ;Конец программы
```


7. Таймер

Важной составной частью микроконтроллера является таймер. Таймер — это синхронизированный счетчик, который по каждому тактовому сигналу считает в прямом или обратном направлении, т. е. считает с увеличением или уменьшением своего значения. В микроконтроллере имеется 3 таймера. Таймеры могут использоваться для самых различных целей. Они нужны, например, для построения сторожевой схемы или для организации цикла ожидания. Кроме того, с помощью таймеров можно определять время между двумя импульсами или формировать импульсы с определенной длительностью. Каждый таймер располагает своим регистром, в котором сохраняется значение счета в прямом или обратном направлении. При переполнении или опустошении этого регистра может вызываться прерывание. Можно также запрограммировать таймер так, что при загрузке в регистр некоторого значения осуществляется в цикле обратный счет, пока его содержимое не станет равно 0. Это часто требуется для простой реализации кратковременной задержки в программе. В приведенном далее примере программы вывод RA2 микроконтроллера устанавливается в состояние высокого логического уровня на время около 100 мкс. В случае если контроллер тактируется частотой 4 МГц, то в регистр COUNTER нужно загрузить десятичное значение 32. В этом случае следует учитывать, что команда `decfsz` выполняется за 1 командный цикл (за время 1 мкс), а команда `goto` — два командных цикла (требуется 2 мкс).

Пример:

<code>bsf PORTA, 2</code>	<code>;Установить вывод RA2 на высокий уровень</code>
<code>movlw D'32'</code>	<code>;Загрузить регистр COUNTER значением 32</code>
<code>movwf COUNTER</code>	
<code>decfsz COUNTER, F</code>	<code>;Уменьшить значение на 1</code>
<code>goto \$-1</code>	<code>;Считать вниз, до тех пор пока значение ;не будет равно 0</code>
<code>bcf PORTA, 2</code>	<code>;Установить на выводе RA2 низкий уровень</code>

Очень короткие задержки, в несколько микросекунд, можно очень хорошо реализовать с помощью нескольких команд пор. Этой команде при тактовой частоте 4 МГц требуется для выполнения один командный цикл, что соответствует времени выполнения 1 мкс. Подобным способом без проблем может осуществляться временная задержка примерно до 765 мкс ($3 * 255$) без применения таймера. Если же потребуется время более 765 мкс, то для аналогичного способа задания времени задержки необходимо использовать уже 2 регистра.

7.1. 8-разрядный таймер (Timer0)

Использование таймера имеет смысл для того, чтобы задавать более длительное время. Большое преимущество таймера состоит в том, что тактовый сигнал, с помощью которого регистр таймера будет осуществлять прямой или обратный счет, поступает с присоединенного *предварительного делителя частоты* или, иначе говоря, *предделителя частоты*. В этом случае значение регистра таймера больше не изменяется с частотой выполнения команд, а с соответствующей более низкой частотой. Коэффициент деления предделителя частоты может выбираться от 1:2 до 1:256. Например, если коэффициент деления предделителя частоты задать равным 1:256 и 8-разрядный таймер загрузить шестнадцатеричным значением 0xFF (255), то это уже будет соответствовать задержке 65280 мкс = 65,28 мс. В следующем примере таймер Timer0 используется для временной задержки на 10 мс.

Пример:

```

_BANK_1                ;Переключиться на Банк 1
movlw b'11000111'      ;Коэффициент деления предделителя частоты
                        ;для таймера Timer0 задать равным 1:256

movwf OPTION_REG

_BANK_0                ;Переключиться на Банк 0
movlw b'00100000'      ;Отключить прерывания и сбросить флаг
movwf INTCON           ;TMR0IF таймера Timer 0 (TMR0) в рег. INTCON
bsf PORTA, 2           ;Установить на выводе RA2 высокий уровень

_BANK_2
movlw D'217'
movwf TMR0             ;Запустить таймер

_BANK_0
btfs INTCON, TMR0IF    ;Проверить, переполнен ли таймер
goto $-1
bcf PORTA, 2           ;Установить на выводе RA2 низкий уровень

```

Сначала в регистре OPTION_REG устанавливается коэффициент деления предделителя (предварительного делителя частоты) на значение 1:256, который

подключается к таймеру Timer0. Этот таймер можно использовать совместно со сторожевым таймером WDT, который входит в его состав, поэтому исключено одновременное его применение Timer0 и как отдельного таймера, и как сторожевого таймера. Далее по программе отключаются все прерывания, но устанавливается бит разрешения прерывания при переполнении таймера Timer0, однако перехода на программу обслуживания прерывания пока не будет. Далее на выводе RA2 устанавливается высокий логический уровень и запускается таймер. Затем в цикле выполняют проверку момента установки флага TMR0IF переполнения таймера, т. е. когда он переходит из состояния 0xFF в 0x00. Перед запуском таймера его надо инициализировать, записав в регистр TMR0 требуемое значение для нужного времени задержки. Поскольку Timer0 при каждом тактовом импульсе будет инкрементирован (осуществляется прямой счет), то значение для инициализации должно рассчитываться по формуле 7.1.

$$TMR0 = 256 - \frac{F_{osc} \cdot t}{4 \cdot PS} \quad (7.1)$$

TMR0 — значение счетчика Timer0;

F_{osc} — частота генератора;

t — временная задержка;

PS — значение делителя (Prescale).

Для достижения временной задержки 10 мс требуется следующее значение:

$$TMR0 = 256 - \frac{4 \text{ МГц} \cdot 10 \text{ мс}}{4 \cdot 256} = 256 - \frac{4 \cdot 10^6 \cdot 10 \cdot 10^{-3}}{4 \cdot 256} = 216,94 \approx 217$$

Это значение и записывалось в примере программы в регистр TMR0 счетчика Timer0. После того как значение будет записано в регистр TMR0, таймер начнет работать, и будет осуществляться процесс ожидания момента установки флага прерывания TMR0IF. После этого сигнал на выводе RA2 снова возвращается на низкий логический уровень, формируя тем самым положительный импульс (высокого уровня) длительностью 10 мс. Поскольку в регистр TMR0 можно записать только целые значения и коэффициент деления предварительного делителя частоты был установлен на значение 1:256, то самая маленькая временная задержка будет составлять 256 мкс.

7.2. 16-разрядный таймер (Timer1)

Модуль таймера Timer1 имеет в своем распоряжении два 8-битных регистра (TMR1L и TMR1H). Этот таймер считает аналогично таймеру Timer0, и может вызвать прерывание, если значения 0xFF в регистрах TMR1L и TMR1H

переходят в состояние 0x0000. С помощью двух 8-разрядных регистров (в общей сложности 16 разрядов) этот таймер может считать до 65535 (0xFFFF), до момента его переполнения. Разумеется, значение коэффициента деления для предделителя этого таймера может устанавливаться не настолько высоким, как для Timer0. Для таймера Timer1 возможен максимальный коэффициент делителя равный 1:8. Это значит, что при тактовом сигнале с частотой 4 МГц возможна максимальная временная задержка, равная примерно 0,524 секунд при временной разрешающей способности 8 мкс. Модуль таймера Timer1 может эксплуатироваться в двух различных режимах работы. Первый режим работы — это применение его как таймера, с помощью которого можно реализовать временную задержку или измерять время между двумя событиями (например, двумя импульсами). Во втором режиме работы можно использовать оба регистра как счетчик и таким образом считать количество внешних импульсов. Поскольку эти импульсы имеют свой определенный период следования, то с их помощью можно реализовать соответствующее время ожидания. Содержимое регистров таймера Timer1 увеличивается при каждом положительном фронте тактового импульса, т. е. при его изменении с состояния логического 0 на 1.

В следующем примере показан программный код для программирования "бегущих огоньков" на светодиодах. Для этого светодиоды включаются по очереди на 0,4 секунды.

Пример:

```
start                                ;Начало программы
;Инициализация
_BANK_0
clrfr PORTA                        ;Очистить все исходные порты ввода/вывода
clrfr PORTB
clrfr PORTC
_BANK_1
movlw b'11111111'                  ;Все выводы порта С настроить в качестве
movwf ADCON1                       ;цифровых входов или выходов
movlw b'11110011'                  ;Сделать выводы RA3 и RA2 выходами,
movwf TRISA                         ;а остальные входами
movlw b'11000000'                  ;Сделать выводы RB7 и RB6 входами,
movwf TRISB                         ;а остальные выходами
movlw b'11111111'                  ;Все выводы порта С настроить входами
movwf TRISC
clrfr INTCON                       ;Отключить прерывания
clrfr PIE1
_BANK_0
clrfr PIR1
```

```

main                                ;Начало основного цикла
;Программный код
;включения светодиода LED 1 на 0,4 секунды
movlw 0x3C                          ;Загрузить значение 15536=0x3CB0 в оба
movwf TMR1H                         ;регистра таймера Timer1
movlw 0xB0
movwf TMR1L
bsf LED_1                           ;Включить светодиод LED 1
movlw b'00110001'                  ;Задать коэфф. деления для предделителя
movwf T1CON                         ;частоты 1:8 и включить таймер Timer1
btfss PIR1, TMR1IF
goto $-1                            ;Ожидание переполнения таймера
bcf LED_1                           ;Выключить светодиод LED 1
clrf PIR1                           ;Сбросить бит переполнения
;Включение светодиода LED 2 на 0,4 секунды
movlw 0x3C                          ;Загрузить значение 15536=0x3CB0 в оба
movwf TMR1H                         ;регистра таймера Timer1
movlw 0xB0
movwf TMR1L
bsf LED_2                           ;Включить светодиод LED 2
movlw b'00110001'                  ;Задать коэфф. деления для предделителя
movwf T1CON                         ;частоты 1:8 и включить таймер Timer1
btfss PIR1, TMR1IF
goto $-1                            ;Ожидание переполнения таймера
bcf LED_2                           ;Выключить светодиод LED 2
clrf PIR1                           ;Сбросить бит переполнения
;Включение светодиода LED 3 на 0,4 секунды
movlw 0x3C                          ;Загрузить значение 15536=0x3CB0 в оба
movwf TMR1H                         ;регистра таймера Timer1
movlw 0xB0
movwf TMR1L
bsf LED_3                           ;Включить светодиод LED 3
movlw b'00110001'                  ;Задать коэфф. деления для предделителя
movwf T1CON                         ;частоты 1:8 и включить таймер Timer1
btfss PIR1, TMR1IF
goto $-1                            ;Ожидание переполнения таймера
bcf LED_3                           ;Выключить светодиод LED 3
clrf PIR1                           ;Сбросить бит переполнения
;Включение светодиода LED 4 на 0,4 секунды
movlw 0x3C                          ;Загрузить значение 15536=0x3CB0 в оба
movwf TMR1H                         ;регистра таймера Timer1
movlw 0xB0
movwf TMR1L

```

```

bsf LED_4           ;Включить светодиод LED 4
movlw b'00110001'   ;Задать коэф. деления для предделителя
movwf T1CON          ;частоты 1:8 и включить таймер Timer1
btfss PIR1, TMR1IF   ;Ожидание переполнения таймера
goto $-1            ;Выключить светодиод LED 4
bcf LED_4           ;Сбросить бит переполнения
clrf PIR1
goto main

```

В начале программы входы и выходы устанавливаются соответственно расположению выводов на монтажной плате. Чтобы вначале при переполнении счетчика не было перехода в программу обслуживания прерывания, все прерывания запрещаются. Далее в программе циклически опрашивается флаг прерывания при переполнении регистра таймера Timer1. Для осуществления временной задержки равной 0,4 секунды при частоте тактового генератора 4 МГц в два регистра таймера Timer1 (TMR1L и TMR1H) должно быть загружено значение 15536. Для определения значения для загрузки в оба регистра таймера можно воспользоваться формулами 7.2, 7.3 и 7.4.

$$TMR = 65536 - \frac{F_{osc} \cdot t}{4 \cdot PS} \quad (7.2)$$

$$TMR1H = \frac{TMR}{256} \quad (7.3)$$

$$TMR1L = TMR - TMR1H \cdot 256 \quad (7.4)$$

TMR — значение регистров 16-битного таймера;

F_{osc} — частота генератора;

t — временная задержка;

PS — значение предделителя (Prescale);

TMR1H — старший байт 16-битного таймера;

TMR1L — младший байт 16-битного таймера.

Для временной задержки на 0,4 секунды значения регистров рассчитываются, как это показано далее:

$$TMR = 65536 - \frac{4 \text{ МГц} \cdot 0,4 \text{ с}}{4 \cdot 8} = 65536 - \frac{4 \cdot 10^6 \cdot 0,4}{4 \cdot 8} = 15536 = 0x3CB0$$

$$TMR1H = \frac{15536}{256} = 60,6875 \approx 60 = 0x3C$$

$$TMR1L = 15536 - 60 \cdot 256 = 178 = 0xB0$$

Перед включением следующего светодиода значение для временной задержки должно вноситься также в регистры таймера, поскольку прямой счет в регистре от значения 0 осуществлялся бы иначе. После включения светодиода и запуска таймера на счет в цикле проверяется, устанавливался ли флаг переполнения таймера. По команде `goto $-1` программный счетчик переходит к предыдущей команде. Символ `$` стоит для указания текущего состояния программного счетчика. Если произошло переполнение счетчика, то команда `goto $-1` пропускается и по следующей команде светодиод выключается. Флаг переполнения таймера `Timer1` также должен сбрасываться, т. к. иначе в следующем цикле сразу произошло бы снова переполнение, независимо от показания счетчика в регистре таймера.

Задержка с помощью макрокоманды для таймера `Timer1`

Временные задержки в программах применяются достаточно часто. Поэтому имеет смысл использовать макрокоманду, с помощью которой можно, задав некоторый параметр, получить определенную временную задержку. В следующем примере программного кода применена макрокоманда, реализующая временную задержку от 20 мкс до 524 мс. Вычисление соответствующих значений для регистров `TMR1L` и `TMR1H` получают с помощью макрокоманды.

Пример:

```
_DELAY_TMR1_US macro usek
variable timer_HL=0
variable timer_H=0
variable timer_L=0
if usek > d'524000'
error "MACRO: Слишком большое значение для макрокоманды _DELAY_TMR1_US!"
endif
if usek < d'20'
error "MACRO: Слишком малое значение для макрокоманды _DELAY_TMR1_US!"
endif
```

;Вычисление содержимого регистра

```
timer_HL = d'65536'-(OSC_FREQ/d'1000000'*usek/d'4'/d'8')
```

;Вычисление старшего байта

```
timer_H = (timer_HL >> d'8')
```

;Вычисление младшего байта

;прибавляется 1, т. к. макрокоманда состоит из нескольких команд

```
timer_L = (timer_HL & 0x00FF)+d'1'
```

```

bcf P1E1, TMR1IE      ;Запретить прерывания при переполнении Timer1
movlw timer_H          ;Загрузить старший байт в регистр TMR1H
movwf TMR1H
movlw timer_L          ;Загрузить младший байт в регистр TMR1L
movwf TMR1L
movlw b'00110001'      ;Задать коэффициент деления для предделителя
movwf T1CON             ;частоты 1:8 и включить таймер Timer1
btfss PIR1, TMR1IF      ;Проверить флаг переполнения
goto $-d'1'
bcf PIR1, TMR1IF        ;Произошло переполнение таймера
                        ;Сбросить бит переполнения

endm

```

Поскольку макрокоманда использует таймер Timer1 с коэффициентом деления предделителя 1:8, то максимальная точность задержки составляет 1/8 такта команды. В начале макроса проверяется, лежат ли указанные значения в диапазоне, который может реализовать таймер Timer1. Если указанное значение будет больше чем 524000 мкс или же меньше чем 20 мкс, то в этом случае будет выведено сообщение об ошибке, соответствующее конкретной ситуации. Затем рассчитываются значения для регистров TMR1L и TMR1H таймера Timer1. Последующие команды на ассемблере аналогичны командам предыдущего примера. В результате программный код с использованием макрокоманды становится простым и более понятным. Таким образом, время ожидания очень легко может быть изменено без необходимости расчета значения регистра вручную. К сожалению, во время выполнения программы уже нельзя изменить время ожидания, поскольку макрокоманда интерпретируется перед непосредственным процессом трансляции и обработки. Макрокоманда является только заменой программного текста, которая может упрощать программирование.

С помощью макрокоманды предыдущий пример можно записать значительно короче, хотя необходимая память программ будет практически такая же.

```

main
;Включение светодиода LED 1 на 0,4 секунды
bsf LED_1
_DELAY_TMR1_US d'400000'
bcf LED_1
;Включение светодиода LED 2 на 0,4 секунды
bsf LED_2
_DELAY_TMR1_US d'400000'
bcf LED_2
;Включение светодиода LED 3 на 0,4 секунды
bsf LED_3

```



```

_DELAY_TMR1_US d'400000'
bcf LED_3
;Включение светодиода LED 4 на 0,4 секунды
bsf LED_4
_DELAY_TMR1_US d'400000'
bcf LED_4
goto main

```

7.3. Модуль таймера Timer2

Модуль таймера Timer2 — это 8-разрядный таймер/счетчик. Разумеется, Timer2 имеет предделитель (англ. Prescaler — предварительный делитель частоты) и выходной делитель (англ. Postscaler). Предварительный делитель частоты может делить тактовую частоту на 1, 4 или 16. После предварительного делителя частоты следует 8-разрядный регистр таймера/счетчика, а вслед за ним — выходной делитель, который делит выходную частоту таймера еще раз и имеет коэффициент деления от 1:1 до 1:16. Выходной делитель может быть программно настроен очень точно, поскольку имеет 16 различных значений с шагом 1, и поэтому возможны такие коэффициенты деления, как 1:3 или 1:13. Модуль таймера Timer2 может быть опорным таймером для первого модуля сравнения/захвата/ШИМ (англ. CCP1) в режиме *шиотно-импульсной модуляции* ШИМ (англ. PWM). Генерация прерывания происходит не как в других таймерах (Timer0 и Timer1) при переполнении регистра таймера, а после сравнения содержимого регистра TMR2 таймера с содержимым специального 8-разрядного регистра PR2. Регистр PR2 — это так называемый *регистр периодов*, с помощью которого можно задавать временной период. Значение в регистре TMR2 таймера (Timer2) может увеличиваться до тех пор, пока оно не будет равно содержимому регистра PR2, после чего значение в регистре TMR2 сбрасывается на 0. Выход компаратора присоединяется к входу выходного делителя и еще раз делится на заданный коэффициент. Для упрощения понимания функционирования модуля таймера можно обратиться к блок-схеме на рис. 7.1.

Для вычисления временной задержки можно использовать формулу 7.5. На основе 8-битного регистра PR2 и максимального двукратного деления частоты на 16 с помощью этого таймера возможна максимальная временная задержка равная примерно 65 мс.

$$PR2 = \frac{F_{osc} \cdot t}{4 \cdot PRE \cdot POST} - 1 \quad (7.5)$$

PR2 — регистр периода;

F_{osc} — частота генератора;

t — временная задержка;

PRE — коэффициент деления предварительного делителя частоты;

POST — коэффициент деления выходного делителя.

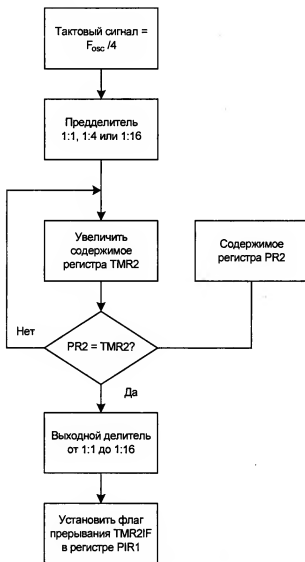


Рис. 7.1. Принцип работы модуля таймера Timer2

Если, например, используется кварц на 4 МГц и требуется установить временную задержку на 1 мс, нужно задать коэффициент деления для выходного делителя 1:10, поскольку при таком значении можно получить самый точный результат. Для предварительного делителя частоты целесообразно установить значение 1:1. Таким образом, подставляя требуемые значения в формулу 7.5, можно получить необходимое содержимое регистра PR2.

$$PR2 = \frac{4 \text{ МГц} \cdot 1 \text{ мс}}{4 \cdot 1 \cdot 10} - 1 = \frac{4 \cdot 10^6 \cdot 1 \cdot 10^{-3}}{4 \cdot 10} - 1 = 99$$

При коэффициентах деления делителей 1:10 и 1:1 получаются целочисленные значения и при этом не возникают никаких ошибок при округлении. В следующем примере программного кода с помощью модуля таймера Timer2 осуществляется временная задержка на 1 мс.

Пример:

```

_BANK_1          ;Переключиться на Банк 1
movlw d'99'       ;Регистр периода PR2 загрузить значением 99
movwf PR2
_BANK_0          ;Переключиться на Банк 0
clrf TMR2
movlw b'01001100' ;Задать коэффициент деления для предделителя
movwf T2CON       ;частоты 1:1, выходного делителя 1:10, Timer2 вкл.
bsf PORTA, 2      ;Установить на выводе RA2 высокий лог. уровень
btfss PIR1, TMR2IF ;Проверить момент установки флага прерывания при
goto $-1          ;соответствии значений в регистрах TMR2 и PR2
bcf PIR1, TMR2IF  ;Сбросить флаг прерывания TMR2IF
bcf PORTA, 2      ;Установить на выводе RA2 низкий лог. уровень

```


8. Обработка аналоговых сигналов

Многие микроконтроллеры могут обрабатывать не только цифровые, но и аналоговые сигналы. Эти сигналы поступают, например, от датчика температуры, который в зависимости от температуры выдает соответствующий аналоговый сигнал. Чтобы сигналы от датчиков могли обрабатываться микроконтроллером, их зачастую надо дополнительно усиливать. При помощи аналогового входа микроконтроллера можно, например, контролировать напряжение батареи и выводить предупреждение при уменьшении напряжения ниже определенного уровня. Аналоговые входы могут использоваться для сигналов, которые изменяются достаточно медленно с частотой примерно до 1—5 кГц. Измерение сигналов, изменяющихся с большей частотой, также возможно, но, разумеется, тогда частота измеряемых сигналов будет сильно зависеть от времени преобразования. Если же помимо измерения выполняются еще и необходимые вычисления, то в этом случае имеет смысл измерять сигналы с частотой около 100 Гц или даже меньше. Исходя из этих ограничений, можно сказать, что микроконтроллер не подходит для обработки стандартных звуковых сигналов с частотой от 20 Гц до 20 кГц. Поэтому для определения максимальной частоты измеряемых аналоговых сигналов важно учитывать не только время преобразования, но так же и время обработки в программе. Если, например, различные значения при измерении записываются во внутреннюю EEPROM-память, то максимальная измерительная частота составляет около 250 Гц, поскольку для сохранения во внутреннюю EEPROM-память требуется минимум 4 мс. Однако температура или напряжение батареи питания, как правило, изменяются достаточно медленно, поэтому без проблем можно осуществлять измерения при секундном или минутном тактовом сигнале.

8.1. Аналого-цифровое преобразование

Поскольку аналоговый сигнал не может быть непосредственно записан в регистр, а поэтому должен быть предварительно преобразован в цифровой код.

Непрерывный аналоговый сигнал можно представить, состоящим из бесконечно большого количества минимальных его изменений, следовательно, для регистрации аналогового сигнала также понадобились бы бесконечно большой регистр и бесконечно большая арифметическая производительность микроконтроллера. Это практически невозможно, а кроме того, не имеет большого смысла. Поэтому аналоговое напряжение разделяют на небольшие ступени. В области между этими ступенями поддерживают определенное цифровое значение. Чем мельче ступени, тем больше точность, а также *разрешающая способность АЦП*, которая напрямую связана с его *разрядностью* и указывается в битах. Разрядность АЦП характеризует количество дискретных значений, выдаваемых преобразователем на выходе. Так АЦП микроконтроллера PIC16F876A может преобразовывать аналоговые сигналы с разрешающей способностью 10 битов. При этом 10 битов АЦП будут определять 2^{10} ступеней, соответствующих 1024 значениям. Следовательно, максимальное измеряемое напряжение может делиться на 1024 маленьких частей, из которых и будет составляться любое цифровое значение. Насколько большой будет каждая такая часть, зависит от *опорного напряжения*, с помощью которого получается цифровое значение. Если максимальное значение аналогового напряжения равно 5 В (рис. 8.1), то при разрешающей способности

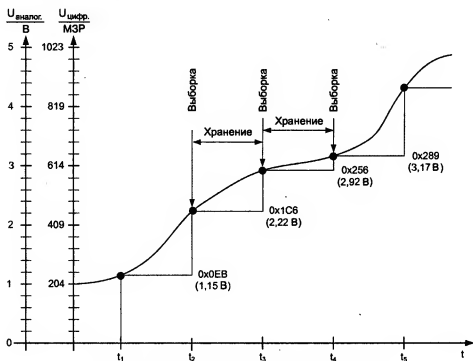


Рис. 8.1. Процесс преобразования аналогового сигнала

10 битов (при 10-разрядном АЦП) соответствующее цифровое значение будет равно 0x3FF. Величина каждой ступени при этом будет примерно равна 4,88 мВ (5 В / 1024), иначе говоря, при повышении аналогового напряжения на 4,88 мВ соответствующее цифровое значение будет изменяться на 1 МЗР (младший значащий разряд) (LSB) или 1 бит.

8.1.1. АЦП-преобразование методом поразрядного уравнивания

Имеются различные способы преобразования аналогового напряжения в цифровое значение, как, например, с помощью сигма-дельта-преобразования (Sigma-Delta) или метода поразрядного уравнивания (последовательных приближений). Преобразование сигма-дельта (Sigma-Delta) выполняется относительно быстро, но в то же время очень дорогое при реализации, поэтому в микроконтроллерах не применяется. В рассматриваемом в этой книге микроконтроллере PIC16F876A используется АЦП поразрядного уравнивания.

На рис. 8.2 представлена его принципиальная схема. Как можно заметить, на входе схемы находится мультиплексор, с помощью которого можно выбрать требуемый вывод в качестве аналогового входа. Поскольку в микроконтроллере имеется только одна схема АЦП, то нельзя одновременно измерить несколько входных напряжений. Поэтому, при необходимости таких измерений, их выполняют по очереди. Это, разумеется, может приводить к ошибкам при быстром изменении аналоговых сигналов. Если, например, измеряется мощность на нагрузке, то сначала должен измеряться ток, а затем напряжение. При измерении тока после считывания электрического сигнала требуется еще некоторое время на осуществление преобразования его аналогового значения в цифровой код. Только после этого может начаться измерение напряжения. Однако в этот момент времени текущее значение напряжения уже может быть изменено и, таким образом, после последующего измерения и умножения в измеряемую в итоге мощность будет внесена ошибка. Выбор аналогового канала осуществляется с помощью трех битов CHS2—CHS0 в регистре ADCON0.

После выбора канала мультиплексором аналоговый сигнал поступает на устройство выборки и хранения (Sample and Hold). При замыкании ключа достаточно быстро заряжается конденсатор хранения. После того как конденсатор будет полностью заряжен, ключ размыкает цепь, и измеряемое напряжение на некоторое время сохраняется на конденсаторе. Это становится возможным, поскольку последующий операционный усилитель (компаратор) имеет очень высокоомный вход, через который протекает ничтожно малый ток. Чтобы конденсатор хранения мог быстро заряжаться, внутреннее сопротив-

ление аналогового источника должно быть по возможности низкоомным. Сопротивление аналогового источника не должно превышать 2,5 кОм. Если же аналоговый источник сигнала имеет более высокое сопротивление, то нужно использовать дополнительный операционный усилитель в качестве согласования. Разумеется, что можно использовать источник и с более высоким сопротивлением без применения дополнительного операционного усилителя, но при этом нужно учитывать, что для заряда конденсатора хранения требуется более длительное время. Найти соответствующую формулу для вычисления можно в техническом паспорте на микроконтроллер.

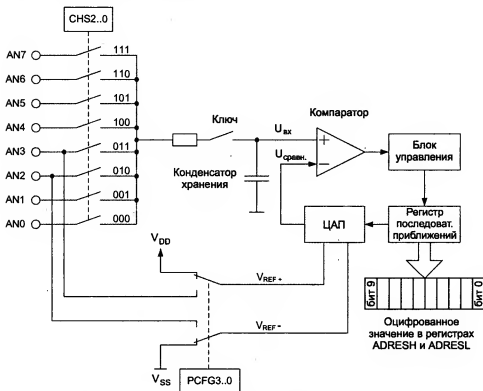


Рис. 8.2. Структурная схема АЦП

За конденсатором выборки и хранения в настоящей схеме следует узел для преобразования аналогового сигнала в цифровой код. В данном микроконтроллере для преобразования аналогового сигнала применяется схема поразрядного уравнивания (последовательных приближений). Следует заметить, что использование в данном случае понятия "превращение" было бы, наверное, более верным, поскольку сам измеряемый сигнал не изменяется, а меняется только его форма представления из аналоговой в цифровую. Однако

понятие "АЦ-превращение" на практике не используется, а применяется понятие "Аналого-цифровой преобразователь — АЦП" (от англ. *Analog-to-Digital Converter* — *ADC*) и соответственно АЦ-преобразование.

При использовании метода поразрядного уравнивания (последовательных приближений) на один вход компаратора подается постоянное измеряемое напряжение $U_{\text{вх}}$ (U_{EIN}). С помощью этого компаратора проверяется, является ли приложенное напряжение больше или меньше напряжения сравнения $U_{\text{сравн.}}$. Напряжение для сравнения формирует вспомогательный цифроаналоговый преобразователь (ЦАП) (англ. *Digital-to-Analog Converter* — *DAC*), который генерирует аналоговый сигнал из цифрового значения регистра последовательных приближений. При этом на ЦАП сначала подается только старший бит и с помощью компаратора проверяется, больше или меньше напряжение $U_{\text{сравн.}}$, полученное с выхода ЦАП, входного измеряемого напряжения $U_{\text{вх}}$. Если входное напряжение меньше чем $U_{\text{сравн.}}$, то на выходе компаратора будет сигнал логического 0. Это значение сохраняется в регистре. На втором шаге на ЦАП подается старший и следующий бит и снова осуществляется проверка, больше или меньше входное напряжение $U_{\text{вх}}$, полученного напряжения с ЦАП. Эта процедура повторяется до тех пор, пока к ЦАП не будут приложены все биты регистра последовательных приближений.

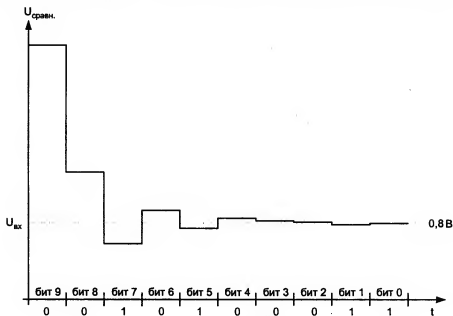


Рис. 8.3. Принцип метода поразрядного уравнивания

В случае с 10-разрядным АЦП это продолжается 10 циклов тактового сигнала. Для иллюстрации принципа преобразования на рис. 8.3 представлен процесс преобразования входного напряжения 0,8 В в 10-разрядном АЦП с поразрядным уравниванием при значении опорного напряжения 5 В.

Из рисунка видно, как в зависимости от шага преобразования, т. е. подключения более младшего бита регистра последовательного приближения, постепенно возрастает точность преобразования. После завершения процедуры преобразования в этом регистре будет цифровое значение 0x0A4, которое и соответствует аналоговому значению 0,8 В.

8.1.2. Передаточная функция АЦП

Какое значение формируется при преобразовании, можно увидеть на рис. 8.4, где представлена передаточная функция АЦП.

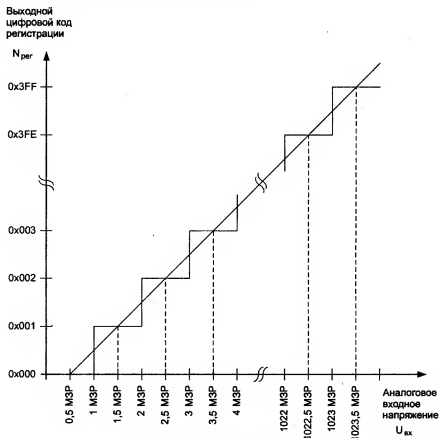


Рис. 8.4. Передаточная функция АЦП

Младший значащий разряд МЗР (LSB) соответствует напряжению 4,883 мВ при опорном напряжении (U_{REF}) 5 В и может рассчитываться по формуле 8.1.

$$1 \text{ МЗР (LSB)} = \frac{U_{REF}}{1024} \quad (8.1)$$

8.1.3. Вычисление значения напряжения

У цифрового значения всегда имеется погрешность в 1 МЗР (LSB). Если аналоговое значение соответствует 2 МЗР (LSB), то нельзя быть уверенным, является ли цифровое значение равным 0x001 или уже 0x002, т. к. именно здесь находится порог срабатывания. Поэтому младший бит всегда является неопределенным. Для расчета выходного цифрового значения кода регистрации N_{per} преобразователя можно использовать формулу 8.2.

$$N_{per} = \left(\frac{U_{вх}}{U_{REF}} \cdot 1024 \right) - 0,5 \quad (8.2)$$

После вычисления выходного цифрового значения кода регистрации оно должно округляться в сторону увеличения или до целочисленного значения. Соответствующим образом аналоговое значение может рассчитываться по формуле 8.3, полученной из формулы 8.2.

$$U_{вх} = \frac{N_{per} + 0,5}{1024} \cdot U_{REF} \quad (8.3)$$

Пример:

Для примера положим, что на вход подается аналоговое напряжение 0,8 В и опорное напряжение, равное рабочему напряжению (5 В).

$$N_{per} = \left(\frac{0,8 \text{ В}}{5 \text{ В}} \cdot 1024 \right) - 0,5 = 163,34 = 163$$

$$U_{вх} = \frac{163 + 0,5}{1024} \cdot 5 \text{ В} = 0,798 \text{ В}$$

При значении выходного цифрового кода регистрации преобразователя равного 163 измеренное аналоговое напряжение находится между 0,796 и 0,801 В. Рассчитанное аналоговое напряжение находится ровно посередине обоих значений (0,798 В). Следовательно, результат:

$$U_{вх} = 0,798 \text{ В} \pm 0,5 \text{ МЗР (LSB)} = 0,798 \text{ В} \pm 2,44 \text{ мВ}$$

Итак, очевидно, что у цифрового значения всегда имеется определенный коэффициент погрешности. Во время отсчета показаний приборов и при дальнейших вычислениях нужно обращать внимание на это допускаемое отклонение. Следует учитывать, что опорное напряжение также оказывает большое влияние на точность цифрового значения. Если точность колеблется около 1%, результат аналого-цифрового преобразования также имеет погрешность около 1%. При этом очень просто использовать рабочее напряжение в качестве опорного напряжения. Разумеется, при аналоговом измерении нужно обращать внимание на то, чтобы питающее напряжение во время выполнения преобразования не нагружалось большим количеством потребителей. В противном случае возможно уменьшение питающего напряжения и вследствие этого будет более низкое опорное напряжение.

Чтобы цифроаналоговый преобразователь (DAC) мог выдавать верное аналоговое напряжение на компаратор, к нему необходимо подключить опорное напряжение. С помощью битов PCFG3—PCFG0 регистра ADCON1, управляющих переключением каналов АЦП и опорного напряжения, к цифроаналоговому преобразователю (DAC) могут быть подключены различные опорные напряжения. Проще всего для этих целей использовать питающее напряжение ($V_{REF+} = V_{DD}$ и $V_{REF-} = V_{SS}$). Кроме этого, к выводам AN2 и AN3 микроконтроллера можно подключать и внешнее опорное напряжение (см. рис. 8.2). Таким образом, имеется возможность подавать любое опорное напряжение, например, равное 2 В. В этом случае при подаче 2-вольтового аналогового входного напряжения на выходе АЦП получится максимальное цифровое значение (0x3FF).

8.1.4. Выравнивание оцифрованного значения

В микроконтроллере PIC16F876A имеются только 8-битные регистры, однако оцифрованное значение состоит из 10 бит. Поэтому для его записи необходимо два регистра или иначе регистровая пара ADRESH и ADRESL. В этом случае имеются 2 возможных варианта для сохранения значения в сдвоенный регистр результата аналого-цифрового преобразования ADRESx (англ. Analog-to-Digital Result), состоящий из 16 битов. Можно записывать оцифрованное значение с выравниванием по левому или правому краю. С помощью бита ADFM в регистре ADCON1 выбирают нужный вариант выравнивания. Как правило, выбор варианта зависит от последующей обработки измерения. Если, например, измеряется напряжение батареи и можно отказаться от обоих младших битов результата, то оцифрованное значение можно записать после выравнивания по левому краю и использовать вообще только регистр ADRESH. Если же к оцифрованному значению будет добавляться константа, то вариант выравнивания по правому краю лучше, поскольку никакие опера-

ции сдвига больше выполнять не нужно. Как можно видеть на рис. 8.5, неиспользованные биты регистров ADRESH и ADRESL наполняются нулями.

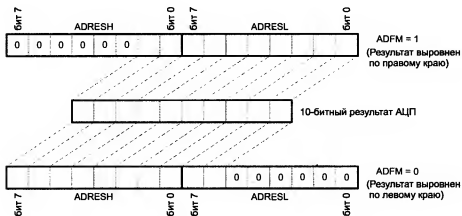


Рис. 8.5. Выравнивание оцифрованного значения

8.2. Пример программы "Вольтметр"

В последующих разделах представлен пример программы, которая управляет светодиодами в зависимости от аналогового входного напряжения. Можно использовать этот пример также для 4-уровневого отображения состояния заряда батареи. Полный работоспособный пример находится на прилагаемом к книге компакт-диске и может имитироваться в среде разработки MPLAB также без использования аппаратных средств. Поскольку здесь речь идет об аналоговых сигналах, то имитация в среде разработки MPLAB несколько затруднена, т. к. симуляция преимущественно предназначена для цифровых сигналов. Однако можно предоставлять для программы не аналоговое значение, а просто перезаписывать содержимое регистров ADRESH и ADRESL нужным значением после аналого-цифрового преобразования. Таким способом можно симулировать остальные части программы. Однако если хотят знать, действительно ли функционирует аналого-цифровое преобразование, то надо собирать схему и испытывать.

Первоначально в программе должны выбираться необходимый порт ввода/вывода и соответствующий вывод в качестве аналогового входа АЦП.

```
start                ;Начало программы
_BANK 0              ;Переключиться на Банк 0
    clrf PORTA        ;Очистить все исходные порты
    clrf PORTB
    clrf PORTC
```

```

_BANK_1                ;Переключиться на Банк 1
movlw b'10001110'      ;Настроить вывод AN0 на аналоговый вход
movwf ADCON1           ;Результат выровнять по правому краю
movlw b'11110011'      ;Настроить выводы RA3 и RA2 в качестве выходов,
movwf TRISA            ;а остальные в качестве входов
movlw b'11000000'      ;Настроить выводы RB7 и RB6 в качестве входов,
movwf TRISB            ;а остальные в качестве выходов
movlw b'11111111'      ;Все выводы порта C сделать входами
movwf TRISC
_BANK_0
movlw b'01000000'      ;Инициализировать АЦП, канал 0 (AN0),
movwf ADCON0           ;Fosc/8 (максимальная тактовая частота 5 МГц)

```

При инициализации АЦП с помощью регистра ADCON1 выбирают, какие выводы используются как аналоговые входы и какие как цифровые входы. В этом случае только вывод AN0 (RA0) выбирают в качестве аналогового входа. Все другие выводы настраивают как цифровые входы, и используют питающее напряжение как опорное напряжение. В этом же регистре задают, что оцифрованное значение напряжения будет сохранено в регистрах ADRESH и ADRESL выровненным по правому краю. Такая настройка была выбрана, поскольку в программе некоторое постоянное значение должно храниться в виде оцифрованного результата. В регистре ADCON0 устанавливается, что аналоговое напряжение должно измеряться в канале 0 (вывод AN0), и что тактовый сигнал для преобразования составляет одну восьмую часть тактовой частоты генератора. Эта настройка важна, поскольку для преобразования 1 бита требуется минимальная длительность 1,6 мкс ($8/5$ мегагерц = 1,6 мкс). Для выбора тактового сигнала для преобразования и определенной тактовой частоты генератора можно воспользоваться табл. 8.1. При изменении настроек нужно обратить внимание на то, что бит ADCS2 находится в регистре ADCON1, а оба бита ADCS1 и ADCS0 должны устанавливаться в регистре ADCON0.

Таблица 8.1. Частота генератора

Тактовый сигнал для преобразования	ADCS2...0	Максимальная тактовая частота генератора
$F_{osc} / 2$	000	1,25 МГц
$F_{osc} / 4$	100	2,5 МГц
$F_{osc} / 8$	001	5 МГц
$F_{osc} / 16$	101	10 МГц
$F_{osc} / 32$	010	20 МГц
$F_{osc} / 64$	110	20 МГц

Теперь регистры для аналого-цифрового преобразования подготовлены и может быть запущен процесс преобразования.

```
main                                ;Начало основного цикла
    _BANK_1
    clrf ADRESL                    ;Очистить младшую часть регистра
                                    ;результата аналого-цифрового преобразования
    _BANK_0
    clrf ADRESH                    ;Очистить старшую часть регистра
                                    ;результата аналого-цифрового преобразования
    bsf ADCON0, ADON               ;Включить модуль АЦП
    bsf ADCON0, GO_DONE            ;Запустить аналого-цифровое преобразование
    btfsc ADCON0, GO_DONE
    goto $-1                       ;Ждать до тех пор, пока аппаратно не будет
                                    ;сброшен бит GO_DONE
    nor                            ;Завершение процесса преобразования
```

В начале основного цикла сначала очищаются регистры для хранения результата аналого-цифрового преобразования ADRESH и ADRESL. Это нужно сделать обязательно, т. к. регистры постоянно снова перезаписываются непосредственно вслед за этим преобразованием. Однако это имеет свое преимущество, поскольку можно быстро заметить, было ли успешно выполнено аналого-цифровое преобразование. Например, если значение 0x0000 после оцифровки все еще находится в регистрах, то с высокой вероятностью была ошибка. Это может быть указанием на то, что было недостаточно времени для преобразования или же, что был заземлен аналоговый вход. Может, конечно, оказаться, что результат будет верен, но только в этом случае, если источник аналогового сигнала будет выдавать очень маленькое напряжение (< 5 мВ).

После очистки регистров включается модуль АЦП и запускается процесс преобразования, в результате установки бита GO_DONE. Этот бит остается в состоянии логической 1 до тех пор, пока процесс преобразования не будет завершен. После завершения процесса преобразования и сохранения результата в соответствующем регистре бит GO_DONE будет сброшен с помощью внутренней аппаратной логики. Команда `nor` вставлена в программу только для установки на ее месте точки останова.

После того как оцифровка аналогового значения будет завершена и результат сохранен в регистре, нужно выполнить проверку того, в каком диапазоне находится полученное значение, для включения надлежащего светодиода. Для этого константа, соответствующая пороговому значению напряжения для определенного уровня диапазона, вычитается из полученного значения и затем проверяется, отрицателен ли результат. Если это действительно так, то изме-

ренное значение напряжения меньше чем пороговое значение, если нет, то больше. Исходя из таких проверок и делается вывод о том, в какой диапазон попадает измеренное входное напряжение. Поскольку оцифрованное значение находится в двух регистрах, то в этом случае нельзя использовать простую команду вычитания (`subwf`). С помощью этой команды может выполняться только 8-битное вычитание, а здесь, однако, требуется 16-битное вычитание. Так как аналогичное действие необходимо во многих программах, то хорошо бы написать рациональную маленькую подпрограмму, которую можно будет затем использовать и в других случаях. Поскольку достаточно часто в программе требуется не только вычитать, но и складывать 16-битные значения, то далее сначала представлен пример 16-битного сложения.

8.3. 16-битное сложение

Чтобы два 16-битных значения, которые состоят из двух 8-битных регистров, можно было складывать, требуются как минимум четыре 8-битных регистра. Для следующей подпрограммы они должны быть определены в начале программного кода и соответствующим образом описаны перед сложением. Речь здесь идет о следующих регистрах:

<code>CALC_L_1</code>	<code>EQU</code>	<code>0x20</code>	;Младшая часть 1-го арифметического регистра ;для сложения и вычитания
<code>CALC_H_1</code>	<code>EQU</code>	<code>0x21</code>	;Старшая часть (слово) регистра <code>CALC_1</code>
<code>CALC_L_2</code>	<code>EQU</code>	<code>0x22</code>	;Второй арифметический регистр <code>CALC_2</code>
<code>CALC_H_2</code>	<code>EQU</code>	<code>0x23</code>	;Пример: <code>CALC_x_1 + CALC_x_2 = CALC_x_1</code> ; <code>CALC_x_1 - CALC_x_2 = CALC_x_1</code>

Эти регистры могут использоваться как для сложения, так и для вычитания. При сложении регистры `CALC_H_1` и `CALC_L_1` складываются с регистрами `CALC_H_2` и `CALC_L_2`, а результат сложения сохраняется опять в регистрах `CALC_H_1` и `CALC_L_1`. Если арифметические регистры были загружены необходимыми значениями, то подпрограмма может вызываться командой `call Add16`.

`Add16`

<code>movf CALC_L_2, W</code>	;Получить младший байт 2-го слагаемого
<code>addwf CALC_L_1, F</code>	;Сложить 1-е и 2-е слагаемое
<code>btfsc STATUS, C</code>	;Проверить, встретилось ли переполнение
<code>incf CALC_H_1, F</code>	;Если да, старший байт увеличить на 1
<code>movf CALC_H_2, W</code>	;Получить старший байт 2-го слагаемого
<code>addwf CALC_H_1, F</code>	;Сложить старшие байты слагаемых
<code>return</code>	;Готово, возврат в основную программу.

После возврата в основную программу биты регистра состояния C, DC и Z будут установлены в зависимости от результата.

8.4. 16-битное вычитание

Вычитание функционирует аналогично сложению. Здесь арифметические регистры CALC_L_1, CALC_H_1, CALC_L_2 и CALC_H_2 также загружаются перед вызовом подпрограммы необходимыми значениями. В подпрограмме содержимое регистров CALC_H_2 и CALC_L_2 вычитается из содержимого соответствующих регистров CALC_H_1 и CALC_L_1.

После описания регистров подпрограмма вызывается при помощи команды call Sub16.

Sub16

```
movf CALC_L_2, W      ;Получить младший байт вычитаемого
subwf CALC_L_1, F      ;CALC_L_1 - CALC_L_2 = CALC_L_1
btfss STATUS, C        ;Встретился ли перенос?
incf CALC_H_2          ;Если да, то старший байт увеличить на 1
movf CALC_H_2, W      ;Получить старший байт вычитаемого
subwf CALC_H_1, F      ;CALC_H_1 - CALC_H_2 = CALC_H_1
return                ;Готово, возврат в основную программу
```

Перед выполнением сложения или вычитания нужно обращать внимание на включение именно того банка памяти, в котором находятся арифметические регистры. В подпрограмме никакая коммутация банков памяти не происходит, т. к. неизвестно, в каком банке определялись регистры для вычисления.

8.5. Анализ оцифрованного значения

На первом шаге анализа проверяется, является ли величина измеренного напряжения меньше 1 В. При опорном напряжении 5 В пороговому напряжению в 1 В для первого уровня соответствует десятичная константа 204 (0x00CC).

```
;Проверить, в каком диапазоне находится оцифрованное значение
movf ADRESH, W        ;Скопировать старшие разряды оцифрованного
                        ;значения в арифметический регистр для вычислений
movwf CALC_H_1
_BANK_1
movf ADRESL, W        ;Скопировать младшие разряды оцифрованного
                        ;значения в арифметический регистр для вычислений
_BANK_0
movwf CALC_L_1
```

```

movlw 0x00          ;Загрузить значение 0x00CC = d'204',
movwf CALC_H_2      ;соответствующее 1 В
movlw 0xCC
movwf CALC_L_2
call Sub16
btfss STATUS, C      ;Проверить, отрицателен ли результат
goto led_aus         ;Если напряжение менее 1 В,
                    ;то все светодиоды отключить

```

Для выполнения проверки арифметические регистры CALC_H_1 и CALC_L_1 для вычислений загружаются оцифрованным значением. При этом нужно обратить внимание, что регистр ADRESL находится в банке 1 и поэтому надо соответствующим образом выполнить переключение, прежде чем содержимое регистра загружается в регистр W. После того как результат аналого-цифрового преобразования был скопирован в 1-й арифметический регистр, значение, соответствующее напряжению в 1 В (0x00CC), загружается во 2-й арифметический регистр. Все необходимые значения теперь загружены в арифметические регистры и может выполняться операция их вычитания посредством вызова подпрограммы командой call Sub16. После вычитания нужно проверить, отрицателен ли результат. Таким образом определяется, меньше или нет измеренное напряжение порогового напряжения равного 1 В для первого уровня. Если меньше, тогда бит переноса C будет иметь значение логического 0 и программа переходит на метку led_aus. В этом случае все светодиоды отключаются, и измерение начинается сначала. Если же бит переноса будет иметь значение логической 1, то измеренное напряжение больше чем 1 В и выполняется следующая проверка, которая определяет, меньше ли это напряжение порогового значения напряжения для второго уровня, т. е. 2 В.

```

;Проверить, находится ли измеренное напряжение между 1 и 2 В
movf ADRESH, W      ;Скопировать старшие разряды оцифрованного
                    ;значения в арифметический регистр для вычислений
movwf CALC_H_1
_BANK_1
movf ADRESL, W      ;Скопировать младшие разряды оцифрованного
                    ;значения в арифметический регистр для вычислений
_BANK_0
movwf CALC_L_1
movlw 0x01          ;Загрузить значение 0x0199 = d'409',
movwf CALC_H_2      ;соответствующее 2 В во 2-й арифметический рег.
movlw 0x99
movwf CALC_L_2
call Sub16

```

```

btfss STATUS, C      ;Проверить, отрицателен ли результат
goto led1_an         ;Если напряжение между 1 и 2 В,
                     ;тогда включить светодиод LED 1

```

Чтобы проверить, находится ли напряжение между 1 и 2 В, значение, соответствующее напряжению 2 В, должно загружаться во второй арифметический регистр. Напряжению 2 В будет соответствовать цифровое значение 0x0199 или 409. Если напряжение находится между 1 и 2 В, результат вычитания положителен и бит переноса будет равен 0. В этом случае программа переходит на метку led1_an для включения 1-го светодиода. Если же измеренное напряжение окажется больше 2 В, то должна выполняться следующая проверка, определяющая, находится ли это значение между 2 и 3 В.

```

;Проверить, находится ли измеренное напряжение между 2 и 3 В
movf ADRESH, W       ;Скопировать старшие разряды оцифрованного
                     ;значения в арифметический регистр для вычислений
movwf CALC_H_1
_BANK_1
movf ADRESL, W        ;Скопировать младшие разряды оцифрованного
                     ;значения в арифметический регистр для вычислений
_BANK_0
movwf CALC_L_1
movlw 0x02            ;Загрузить значение 0x0266 = d'614',
                     ;соответствующее 3 В, во 2-й арифметический рег.
movwf CALC_H_2
movlw 0x66
movwf CALC_L_2
call Sub16
btfss STATUS, C      ;Проверить, отрицателен ли результат
goto led2_an         ;Если напряжение между 2 и 3 В,
                     ;тогда включить светодиод LED 2

```

Для проверки десятичное значение 614, соответствующее уже 3 В, загружается во 2-й арифметический регистр. Оцифрованное значение после аналого-цифрового преобразования, которое находится в регистрах результата ADRESH и ADRESL, должно перед каждой проверкой копироваться в 1-й арифметический регистр вновь, поскольку значение в нем каждый раз перезаписывается после выполнения операции вычитания. Если оцифрованное значение будет меньше 0x0266 (3 В), то программный счетчик переходит на метку led2_an для включения светодиода LED 2. Все другие светодиоды отключаются. Если значение окажется больше 3 В, то требуется выполнить следующую проверку, определяющую, находится ли это значение между 3 и 4 В.

```

;Проверить, находится ли измеренное напряжение между 3 и 4 В
movf ADRESH, W       ;Скопировать старшие разряды оцифрованного
                     ;значения в арифметический регистр для вычислений

```

```

movwf CALC_H_1
_BANK_1
movf ADRESL, W      ;Скопировать младшие разряды оцифрованного
                    ;значения в арифметический регистр для вычислений
_BANK_0
movwf CALC_L_1
movlw 0x03           ;Загрузить значение 0x0333 = d'819',
movwf CALC_H_2       ;соответствующее 4 В, во 2-й арифметический рег.
movlw 0x33
movwf CALC_L_2
call Sub16
btfss STATUS, C      ;Проверить, отрицателен ли результат
goto led3_an         ;Если напряжение между 3 и 4 В,
                    ;тогда включить светодиод LED 3
goto led4_an         ;Если напряжение более 4 В,
                    ;тогда включить светодиод LED 4

```

Если после проверки с помощью операции 16-битного вычитания получилось, что оцифрованное значение находится между 3 и 4 В, то выполняется переход на метку `led3_an` для включения 3-го светодиода. Если же измеренное значение окажется больше 4 В, то никакая дополнительная проверка не требуется, поскольку в этом случае значение может находиться только лишь между 4 и 5 В, и поэтому включается светодиод LED 4.

Для включения или выключения светодиодов устанавливается или сбрасывается соответствующий бит в регистре порта А (PORTA). Для более простого и понятного кода в начале программы сделаны следующие описания:

```

#define LED_1  PORTA, 2  ;LED 1 подключен к выводу RA2
#define LED_2  PORTA, 3  ;LED 2 подключен к выводу RA3
#define LED_3  PORTA, 4  ;LED 3 подключен к выводу RB4
#define LED_4  PORTA, 5  ;LED 4 подключен к выводу RB5

```

После этих описаний светодиод можно включать при помощи команды `bsf LED_x` и соответственно выключать — `bcf LED_x`.

```

led_aus              ;Все светодиоды выключить и
    bcf LED_1        ;снова выполнить измерение
    bcf LED_2
    bcf LED_3
    bcf LED_4
    goto main
led1_an              ;Включить только светодиод LED 1 и
    bsf LED_1        ;снова выполнить измерение
    bcf LED_2

```

```
    bcf LED_3
    bcf LED_4
    goto main
led2_an          ;Включить только светодиод LED 2 и
    bcf LED_1    ;снова выполнить измерение
    bsf LED_2
    bcf LED_3
    bcf LED_4
    goto main
Led3_an          ;Включить только светодиод LED 3 и
    bcf LED_1    ;снова выполнить измерение
    bcf LED_2
    bsf LED_3
    bcf LED_4
    goto main
Led4_an          ;Включить только светодиод LED 4 и
    bcf LED_1    ;снова выполнить измерение
    bcf LED_2
    bcf LED_3
    bsf LED_4
    goto main
```

Аналогичным образом очень просто выполняются и другие световые эффекты. Например, возможен вариант программы, при котором младшие светодиоды также светятся, если напряжение возрастает.

Если же запускают программу на реальных аппаратных средствах, то можно легко заметить, что переходы через пороговые значения будут не очень точными. Если с помощью потенциометра медленно приближаться к пороговому значению напряжения, то будет заметно мерцание двух соседних светодиодов. Это происходит из-за погрешности младших битов и вызывается помехами или непостоянством опорного напряжения.

9. Отображение данных на индикаторе

Во многих случаях данные, находящиеся внутри микроконтроллера, должны сообщаться пользователю устройства. В простейших случаях это можно сделать с помощью светодиода, который показывает, например, превышение тех или иных данных некоторого порогового значения. Если же хотят знать, насколько превышено это пороговое значение, то такое отображение результата посредством светодиодов очень накладно или непрактично. С помощью *жидкокристаллического индикатора* (ЖКИ) или иначе *дисплея* можно вывести результаты вычислений, необходимых пользователю, разными способами. На индикаторе можно представлять самые разнообразные символы. Жидкокристаллические индикаторы изготавливаются всевозможных размеров для самых разных приложений, начиная от маленьких индикаторов, которые могут отображать только 8 символов, до комплексных графических индикаторов, которые могут представлять любые изображения и шрифты в цвете. Поскольку цветные графические индикаторы используются, как правило, чтобы представлять комплексные связанные значения, то для управления такими индикаторами требуются также более мощные микроконтроллеры с достаточным объемом памяти. Небольшие PIC-микроконтроллеры не подходят для управления такими цветными индикаторами и поэтому используются только для управления относительно простыми жидкокристаллическими индикаторами. Как правило, обычно вполне достаточно индикатора для отображения 2×16 символов. На первый взгляд может показаться, что этого очень мало, но удивительно, сколько сведений вместе с тем можно отобразить с помощью даже такого небольшого индикатора. Далее объясняется принцип его работы и управление им. Следует заметить, что управление другими типами индикаторов осуществляется аналогично, и поэтому может выполняться без проблем с помощью приведенных примеров программного кода.

9.1. Контроллер индикатора

На представленной монтажной плате применен жидкокристаллический индикатор серии EA DOG, выпускаемый компанией *Electronic Assembly*. Выбор

пал на эту фирму, поскольку у нее можно приобрести индикатор с фоновой подсветкой. В этом случае имеется преимущество в том, что по желанию можно настраивать тон отображения текста. Кроме того, к этим индикаторам имеется инструкция на немецком языке. Жидкокристаллический индикатор (ЖКИ) имеет 2 строки с 16 символами в каждой. Любой символ может состоять из 5 точек в горизонтальном направлении и 8 точек в вертикальном направлении. В сумме индикатор обеспечивает отображение более $2 \times 16 \times 5 \times 8 = 1280$ точек. Все эти точки, называемые также *пикселями* (Pixel), должны управляться в отдельности. Чтобы управлять в отдельности каждым пикселем, потребуется 1280 линий. Однако PIC-контроллер располагает только 22 выводами ввода/вывода, которых, понятно, недостаточно для управления даже одним символом. Тем не менее, для управления таким ЖКИ производитель использует его совместно со своим внутренним контроллером. Таким образом, с помощью малого количества управляющих линий становится возможным обратиться к контроллеру, который и переключает пиксели в соответствии с переданной ему командой. Поскольку каждый производитель индикаторов использует собственный *контроллер индикатора*, то управление им тоже может быть разным. Так индикаторы компании Electronic Assembly работают под управлением контроллера ST7036. В паспорте компании Electronic Assembly содержится только небольшая часть описания параметров контроллера ST7036. Если же требуются более подробные сведения об управлении, то можно получить их из паспорта на контроллер ST7036 производства компании *Sitronix*.

9.1.1. Набор символов

Поскольку каждый отдельный символ может состоять из 40 точек, то возможно 2^{40} различных комбинаций (более 1 биллиона). Однако немецкий алфавит состоит только из 26 латинских символов, 10 цифр и нескольких умлаутов и специальных символов. Поэтому для вывода пользователю того или иного сообщения требуется не так уж много различных символов. Они предварительно определены и хранятся в контроллере жидкокристаллического индикатора. Вследствие этого также не следует передавать все 40 битов, которые нужны для отдельного символа, а только 8. Вместе с тем могут отображаться 256 различных символов, а для передачи 32 символов требуются 32 байта. Наборы символов для разных контроллеров могут отличаться друг от друга. Это относится, прежде всего, к специальным символам, например, таким как символ "Ом" (Ω) или "микро" (μ). Буквы и цифры в большинстве случаев кодируются так, что они соответствуют набору символов ASCII. На рис. 9.1 представлена только часть набора символов. Полный набор символов можно взять из технического описания ЖКИ, которое находится на прилагаемом к книге компакт-диске.

b7—b4 b3—b0	0000	0001	0010	0011	0100	0101	0110	0111	1000
0000	T	M		Q	Q	P	^	P	S
0001	J	t	!	1	A	Q	a	q	G
0010	w	s	"	2	B	R	b	r	e
0011	F	n	#	3	C	S	c	s	a
0100	d	r	\$	4	D	T	d	t	a
0101	↑	4	%	5	E	U	e	u	a

Рис. 9.1. Часть набора символов жидкокристаллического индикатора

Чтобы отобразить необходимый символ на индикаторе из этого набора символов, нужно выбрать соответствующий символ с помощью 8-битного слова. Так как таблица разделена на столбцы и строки, 4 старших бита (b7—b4) стоят в верхней строке и 4 младших бита (b3—b0) в левом столбце. Чтобы отобразить, например, букву "А", должен указываться двоичный код 0100 0001 и для цифры "5" — 8-битное слово 0011 0101. Эти предварительно определенные символы заданы в памяти контроллера индикатора. Когда и где эти символы должны отображаться, сообщается контроллеру индикатора с помощью специальных команд. Поэтому контроллер должен делать различие между командами управления и командами вывода данных. С помощью управляющих команд контроллеру сообщается, на какое место должен быть установлен символ или о том, что должен очищаться весь индикатор. Данные или символы пишутся во внутреннюю оперативную память (RAM-память) контроллера индикатора. Каждая ячейка памяти содержит символ. Поскольку контроллер индикатора используется также для ЖКИ, у которых имеется и более 2×16 символов, то внутренняя оперативная память соответственно имеет больший размер. При последовательной записи 32 символов во внутреннюю память представляются только первые 16 символов. Поэтому для второй строки должен выбираться наиболее старший адрес. На рис. 9.2 показано, какие адреса памяти должны использоваться для отдельных символов.

Символ	1	2	3	4	13	14	15	16
Строка 1	0x00	0x01	0x02	0x03	0x0C	0x0D	0x0E	0x0F
Строка 2	0x40	0x41	0x42	0x43	0x4C	0x4D	0x4E	0x4F

Рис. 9.2. Адреса памяти жидкокристаллического индикатора

9.1.2. Способы управления индикатором

Следует иметь в виду, что, прежде чем отобразить символы на ЖКИ, сначала надо инициализировать контроллер индикатора. В процессе инициализации в индикатор сообщается, в том числе, насколько должен быть высоким его контраст. Для инициализации необходимы различные команды, которые должны выдаваться PIC-контроллером. Чтобы PIC-контроллер смог обмениваться сообщениями с ЖКИ, необходимо соответствующим образом выполнить подключение аппаратных средств. Для реализации этого имеется несколько возможностей.

Самое простое и самое быстрое управление осуществляется по шине данных шириной 8 бит и трем линиям управляющих сигналов (рис. 9.3). При этом данные могут передаваться к ЖКИ параллельно с применением соответствующей синхронизации. Разумеется, для этого способа управления требуется большое количество линий, которые нельзя будет использовать для других целей.

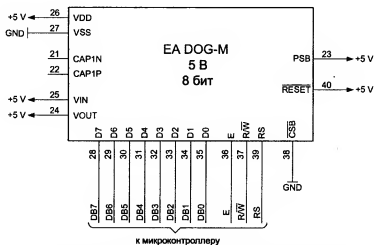


Рис. 9.3. 8-битное управление жидкокристаллическим индикатором

При 4-битном управлении используется меньшее количество линий. В этом случае любая 8-битная команда передается по 4 бита за два раза. Для передачи требуются 2 такта, вследствие чего длительность передачи увеличивается вдвое. Как можно видеть на рис. 9.4, неиспользуемые выходы битов данных D0—D3 подключены к напряжению высокого логического уровня.

Для управления ЖКИ посредством последовательного интерфейса периферийных устройств SPI (Serial Peripheral Interface) вполне достаточно только 4 линий (рис. 9.5). При этом отдельные биты данных передаются на индика-

тор по очереди. Поскольку содержимое ЖКИ, как правило, не меняется с большой скоростью, поэтому такой способ управления является достаточно быстрым и вполне удобным. Именно такое управление используется на монтажной плате, а также поясняется при рассмотрении примеров программного кода.

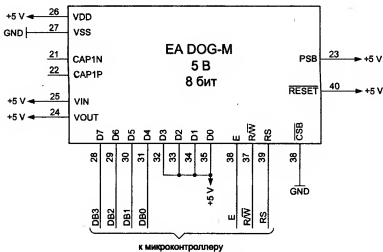


Рис. 9.4. 4-битное управление жидкокристаллическим индикатором

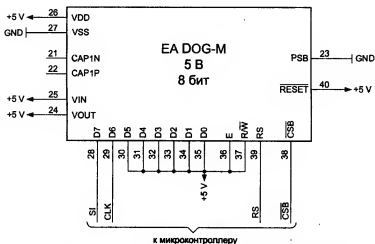


Рис. 9.5. SPI-управление жидкокристаллическим индикатором

9.2. Инициализация индикатора

С помощью сигналов передаваемых на вывод RS осуществляют разделение данных и команд. При наличии низкого логического уровня на этом выводе сигналы на выводе SI интерпретируются как управляющая команда, а при высоком логическом уровне определяются как данные. При передаче данных и команд посредством интерфейса SPI информация на выводе SI воспринимается по положительному фронту тактового сигнала CLK. Для сообщения ЖКИ, что передаваемая информация предназначена именно для контроллера индикатора, сигнал #CSB должен иметь низкий логический уровень. Чтобы отобразить символ "F" в первой строке и в 6-й позиции индикатора, нужно послать сначала команду для выбора соответствующей ячейки памяти, а затем символ "F". Для установки адреса оперативной памяти 0x05, определяющего 6-ю позицию в первой строке индикатора (см. рис. 9.2), нужно передать команду 0x85. Код, соответствующий выводимому символу "F", равен 0x46. Последовательность сигналов для указанной передачи команд и данных по протоколу SPI представлена на рис. 9.6.

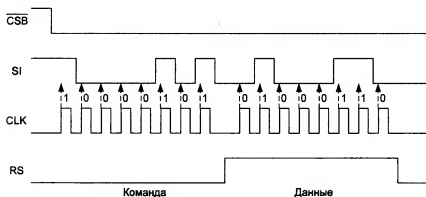


Рис. 9.6. Пример передачи по протоколу SPI

Однако прежде чем отображать символы на ЖКИ, контроллер индикатора при помощи некоторой последовательности специальных команд должен быть проинициализирован. При инициализации, в том числе, ему сообщается, какие линии будут использованы для передачи данных. Чтобы индикатор смог принимать последовательные данные, на вывод #PSB, как это показано на рис. 9.6, должен быть подан сигнал низкого логического уровня. При выполнении инициализации индикатора между командами необходимо вставлять различное время ожидания. Пример алгоритма инициализации 2×16-символьного ЖКИ показан на рис. 9.7. Представленные команды передаются в контроллер индикатора с учетом соответствующего значения для времени

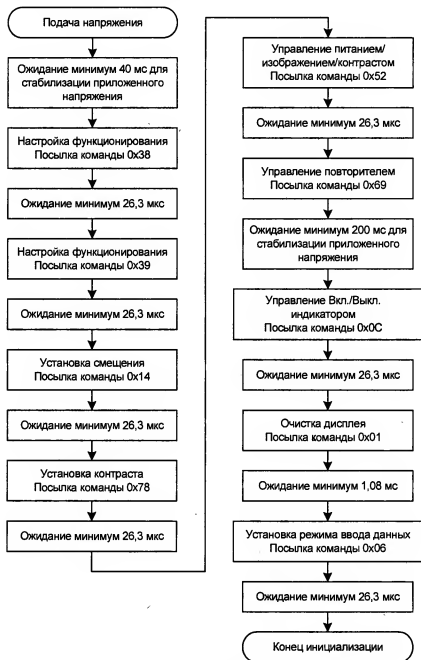


Рис. 9.7. Инициализация индикатора

ожидания. При выполнении инициализации вывод RS все время поддерживается на низком логическом уровне, поскольку при этом передаются только команды. Приведенный пример инициализации для других требований должен соответствующим образом подстраиваться. Подробное описание команд можно найти в документации контроллера индикатора ST7036.

Сначала посредством команд для настройки режима функционирования (Function Set) контроллеру индикатора сообщают, что индикатор должен работать в 2-строчном режиме с нормальной величиной шрифта. Далее командой установки смещения (Bias Set) определяют схему для задания напряжения смещения с помощью делителя из 5 (смещение 1/5) или 4 (смещение 1/4) внешних резисторов. Затем двумя командами установки контраста (Contrast Set) и следующей за ней командой (Power/ICON Control/Contrast Set) с помощью четырех младших (C3—C0) и соответственно двух старших разрядов (C5—C4) настраивают контраст индикатора. В этом примере установленное значение контраста будет равно 0x28. Вслед за этим используется команда управления повторителем (Follower Control), с помощью которой включается внутренний повторитель напряжения. После его включения задают примерно 200 мс времени ожидания для осуществления необходимой стабилизации напряжения. Следующей командой включают индикатор. Для завершения инициализации содержимое ЖКИ очищают и устанавливают, что курсор будет передвигаться вправо. После завершения инициализации данные могут быть записаны непосредственно в оперативную память индикатора (DDRAM—Display Data RAM). Переданные символы сразу же будут отображаться без какой-либо дополнительной команды контроллера.

Если индикатор принял символ полностью, то внутренний указатель автоматически перейдет на последующую ячейку памяти со старшим адресом, и далее будет передаваться символ, записанный именно в эту ячейку. Таким образом, для передачи нескольких символов достаточно указывать только начальную позицию или начальный адрес и после этого посылать желаемое количество символов последовательно на индикатор. Разумеется, при этом нужно быть очень внимательным, поскольку первый символ во второй строке имеет значительно больший адрес, чем адрес последнего символа первой строки (см. рис. 9.2). Поэтому для вывода второй строки символов обязательно указывают новый начальный адрес.

9.3. Интерфейс аппаратных средств

Как уже было сказано ранее, в последующих примерах для передачи данных будет использоваться последовательный интерфейс SPI, поскольку для него требуется небольшое количество линий. Для удобства передачи данных контроллер PIC16F876A имеет встроенный механизм поддержки интерфейса

SPI. Поэтому больше не нужно ломать голову над синхронизацией, а можно предоставить управление передачей PIC-контроллеру. После выполнения передачи данных PIC-контроллер устанавливает специальный флаг, который сообщает пользователю об успешном завершении процедуры. К сожалению, нельзя одновременно использовать оба интерфейса SPI и I²C, поскольку для их реализации нужны одни и те же выводы микроконтроллера. На монтажной плате предусмотрена EEPROM-память, которая передает и принимает данные через интерфейс I²C. Нужно теперь только решить, какой интерфейс должен реализоваться с помощью внутренних аппаратных средств PIC-контроллера. На монтажной плате используется модуль для передачи данных посредством интерфейса I²C, т. к. этот протокол сложнее, чем протокол SPI. Тем не менее, чтобы управлять индикатором через интерфейс SPI, посредством программного обеспечения интерфейс SPI имитируется через четыре цифровых вывода ввода/вывода.

После инициализации индикатора на контроллер индикатора должны посылаться только два вида команд. С одной стороны, это передача команд, например, для выбора адреса оперативной памяти индикатора, а с другой стороны — передача символов. Поскольку эти обе команды очень часто применяются для передачи данных, то для их использования имеет смысл написать две отдельные подпрограммы. Подпрограммы, в противоположность макрокомандам, только однократно занимают место в памяти программ. Так как инициализация индикатора делается только раз в начале программы, то использование небольшой макрокоманды помогает улучшить наглядность программы. Приведенные далее макрокоманда и обе подпрограммы могут затем с успехом использоваться в последующем примере программы.

9.3.1. Подпрограмма для передачи команды

Следующая подпрограмма позволяет передавать команды в контроллер индикатора. Для подпрограммы требуются два регистра. Для их использования в программе они, прежде всего, должны быть определены с помощью директивы EQU.

```
DISP_SHIFT_REG EQU 0x20 ;Регистр для хранения символа или команды
DISP_SHIFT_CNT EQU 0x21 ;Регистр счетчика для вывода данных
```

Итак, данные, предназначенные для передачи, будут храниться в 8-разрядном регистре, но ведь эти данные должны передаваться на ЖКИ последовательно. Поэтому нужна подпрограмма, с помощью которой должно выполняться преобразование в последовательный поток данных. Для этого 8-битное слово сдвигается влево за 8 раз. При сдвиге влево старший бит данных выдвигается на место флага переноса и выводится на последовательный информацион-

ный вывод SI контроллера индикатора. В этом примере рассматривается подпрограмма `SchreibeDispKommando`, которая вызывается по команде `call SchreibeDispKommando`. При этом надо учитывать, что передаваемые данные перед вызовом подпрограммы должны быть загружены в регистр W.

```
SchreibeDispKommando
    movwf DISP_SHIFT_REG      ;Переместить данные в регистр,
                                ;предназначенный для вывода их на индикатор
    bcf DISP_CSB              ;На выводе #CSB установить низкий уровень,
                                ;чтобы индикатор мог принимать информацию
    bcf DISP_RS                ;На выводе RS установить низкий уровень,
                                ;поскольку в данном случае передают команду
    bcf DISP_CLK              ;Установить тактовый сигнал CLK на низкий
                                ;уровень, чтобы данные могли приниматься
                                ;только при возрастающем фронте сигнала
    movlw d'8'                ;Загрузить счетчик количеством битов,
    movwf DISP_SHIFT_CNT      ;предназначенных для передачи (=8)
shiftKommando
    rlf DISP_SHIFT_REG, F      ;Выполнить сдвиг битов команды влево
                                ;Бит переноса выводится на вывод SI ЖКИ
    btfsc STATUS, C           ;Если флаг переноса 0, установить на
                                ;выводе индикатора SI сигнал низкого уровня
    bsf DISP_SI                ;Если флаг переноса равен 1, установить на
                                ;выводе индикатора SI сигнал высокого уровня
    btfss STATUS, C
    bcf DISP_SI
    pop                        ;Задать небольшое время ожидания для
                                ;настройки скорости передачи в индикатор
    bsf DISP_CLK              ;Выдать положительный фронт сигнала CLK
                                ;для приема данных индикатором
    pop
    bcf DISP_CLK              ;Выдать отрицательный фронт сигнала CLK
    decfsz DISP_SHIFT_CNT, F  ;Счетчик битов уменьшить на 1
    goto shiftKommando        ;Если 8 битов еще не выведено,
                                ;перейти на метку 'shiftKommando'
    bcf DISP_SI                ;Если все данные переданы, то
                                ;на выводе SI задать низкий лог. уровень
    bcf DISP_RS                ;Установить на выводе RS низкий уровень
    _DELAY_TMR1_US d'30'      ;Задать время ожидания для индикатора
                                ;(минимум 26,3 мкс)
    return                    ;Возврат из подпрограммы
```

После вызова подпрограммы содержимое регистра W копируется в регистр `DISP_SHIFT_REG`. Затем формируют сигналы `#CSB`, `RS` и `CLK` низкого ло-

гического уровня. Это делается для того, чтобы иметь определенное условие для начала запуска процесса передачи. Для осуществления передачи 8 битов команды в регистр DISP_SHIFT_CNT счетчика записывают 8. Теперь вся подготовка закончена, и данные могут циклически сдвигаться влево, так что старший бит будет поступать на место флага переноса. С помощью команд `btfs STATUS, C` и `btfs STATUS, C` проверяется, имеет ли флаг переноса значение 0 или 1. Проверка должна происходить дважды, поскольку следующая команда, стоящая после команды проверки, в случае выполнения условия пропускается. На выводе индикатора SI теперь будут присутствовать достоверные данные, которые при следующем положительном (возрастающем) фронте тактового сигнала CLK будут приняты индикатором. Затем после некоторой задержки тактовый сигнал можно вернуть на низкий логический уровень, чтобы при выдаче следующего бита информации можно было снова формировать положительный фронт тактового импульса. Теперь после успешного вывода бита информации счетчик DISP_SHIFT_CNT уменьшается на 1. Если все 8 битов будут переданы, то все управляющие сигналы снова возвращаются в исходное состояние, и выдерживается нужное время ожидания 26,3 мкс (в данном случае было выбрано 30 мкс). После этого подпрограмма будет выполнена и осуществляется возврат в основную программу.

9.3.2. Подпрограмма для передачи символа

Подпрограмма для передачи символа работает аналогично подпрограмме для передачи команды. Подпрограммы отличаются только уровнем сигнала на выводе RS, отвечающего за различие команд и данных.

SchreibeDispDaten

<code>movwf DISP_SHIFT_REG</code>	;Переместить данные в регистр,
	;предназначенный для вывода их на индикатор
<code>bcf DISP_CSB</code>	;На выводе #CSB установить низкий уровень,
	;чтобы индикатор мог принимать информацию
<code>bsf DISP_RS</code>	;На выводе RS установить высокий уровень,
	;поскольку в данном случае передают символ
<code>bcf DISP_CLK</code>	;Установить тактовый сигнал CLK на низкий
	;уровень, чтобы данные могли приниматься
	;только при возрастающем фронте сигнала
<code>movlw d'8'</code>	;Загрузить счетчик количеством битов,
<code>movwf DISP_SHIFT_CNT</code>	;предназначенных для передачи (=8)
<code>shiftDaten</code>	
<code>rlf DISP_SHIFT_REG, F</code>	;Выполнить сдвиг битов символа влево
	;Бит переноса выводится на вывод SI ЖКИ
<code>btfs STATUS, C</code>	;Если флаг переноса 0, установить на
	;выводе индикатора SI сигнал низкого уровня

```

bsf DISP_SI          ;Если флаг переноса равен 1, установить на
                     ;выводе индикатора SI сигнал высокого уровня

btfss STATUS, C
bcf DISP_SI
nop                  ;Задать небольшое время ожидания для
                     ;настройки скорости передачи в индикатор

bsf DISP_CLK          ;Выдать положительный фронт сигнала CLK
                     ;для приема данных индикатором

nop                  ;Задать небольшое время ожидания
bcf DISP_CLK          ;Выдать отрицательный фронт сигнала CLK
decfsz DISP_SHIFT_CNT, F ;Счетчик битов уменьшить на 1
goto shiftDaten      ;Если 8 битов еще не выведено,
                     ;перейти на метку 'shiftDaten'

bcf DISP_RS           ;Если все данные переданы, то
                     ;на выводе RS установить низкий уровень

bcf DISP_SI           ;Установить на выводе SI низкий уровень
_DELAY_TMR1_US d'30' ;Задать время ожидания для индикатора
                     ;(минимум 26,3 мкс)

return               ;Возврат из подпрограммы

```

Чтобы сообщить индикатору, что в данном случае осуществляется передача битов символа (данных), на сигнальном выводе RS должен устанавливаться высокий логический уровень.

9.3.3. Макрокоманда для инициализации индикатора

В следующем примере представленная макрокоманда выполняет инициализацию индикатора EA DOG-M компании Electronic Assembly для отображения 2×16 символов. Команды инициализации соответствуют блок-схеме, приведенной на рис. 9.7. Данная макрокоманда может использоваться во всех других программах. Предпосылкой использования макрокоманды, а не подпрограммы является то, что в случае макрокоманды применяется только лишь ее подмена на последовательность заранее определенных команд.

Пример:

```

_INIT_DISPLAY macro      ;Инициализация индикатора EA DOG-M
    movlw 0x38            ;Команда: Function Set
    call SchreibeDispKommando
    movlw 0x39            ;Команда: Function Set
    call SchreibeDispKommando ; (должна передаваться второй раз)
    movlw 0x14            ;Команда: Bias Set
    call SchreibeDispKommando

```

```

movlw 0x78                ;Команда: Contrast Set
call SchreibeDispKommando
movlw 0x52                ;Команда: Power/ICON Control/Contrast Set
call SchreibeDispKommando
movlw 0x69                ;Команда: Follower Control
call SchreibeDispKommando
_DELAY_TMR1_US d'200000' ;Задать время ожидания 200 мс
movlw 0x0C                ;Команда: Display ON/OFF Control
call SchreibeDispKommando
movlw 0x01                ;Команда: Clear Display
call SchreibeDispKommando
_DELAY_TMR1_US d'1100'   ;Задать время ожидания 1,08 мс
movlw 0x06                ;Команда: Entry Mode Set
call SchreibeDispKommando
endm                      ;Конец инициализации

```

Представленная макрокоманда по очереди передает команды инициализации на индикатор. Чтобы исследовать макрокоманду или выполнить ее, имеются две возможности. С одной стороны, можно выполнять код пошагово после дизассемблирования. К сожалению, после дизассемблирования никакие комментарии не сохраняются, так что анализ кода несколько затрудняется. Другая возможность — это временно скопировать код макрокоманды на место обращения к ней и затем запускать на выполнение. Обращение к макрокоманде `_INIT_DISPLAY` (макровывзов) в этом случае должно естественно снабжаться соответствующим комментарием.

9.4. Пример программы "Hello World"

Теперь вся подготовка для отображения текста на ЖКИ закончена. Следующий пример представляет собой классический пример для языка программирования. Его можно найти почти в каждом языке программирования. Здесь речь идет о самой простой возможности вывода текста. С помощью приведенной далее программы на индикаторе будет выведен текст "Hello World!" (Здравствуй мир!) и сразу же будет понятно, была ли успешной передача. В этом примере слово "Hello" отображается в первой строке, начиная с позиции 1, а слово "World!" во второй строке с позиции 8.

Пример:

```

bsf DISP_CSB              ;Подать на индикатор сигнал #CSB высокого
                           ;логического уровня -> ЖКИ пассивен
bcf DISP_CLK              ;Установить сигнал CLK низкого уровня
bcf DISP_SI               ;Установить сигнал SI низкого уровня

```

```

bcf DISP_RS           ;На вывод RS передать низкий уровень,
                        ;соответствующий передаче команд
__INIT_DISPLAY        ;После инициализации содержимое
                        ;индикатора очистить, или иначе
                        ;заполнить его пробелами
main                  ;Начало основного цикла
    movlw 0x80         ;Задать вывод символов в 1-й строке
                        ;с 1-й позиции (=0x80 + 0x00)
    call SchreibeDispKommando ;Передать команду для выполнения
                        ;установки адреса памяти индикатора
    movlw A'H'         ;Загрузить символ "H" в регистр W для
                        ;вывода в 1-й строке с 1-й позиции
    call SchreibeDispDaten ;Выполнить передачу символа
    movlw A'e'         ;Загрузить символ "e" в регистр W
    call SchreibeDispDaten ;Выполнить передачу символа
                        ;с автоматическим увеличением адреса
                        ;для вывода в 1-й строке во 2-ю позицию
    movlw A'l'         ;Загрузить символ "l" в регистр W
    call SchreibeDispDaten ;Передать символ "l" на индикатор
    movlw A'l'         ;в следующую позицию строки
    call SchreibeDispDaten ;Передать еще один символ "l"
    movlw A'o'         ;Загрузить символ "o" в регистр W
    call SchreibeDispDaten ;Вывести последнюю букву в 1-й строке
    movlw 0xC7         ;Задать вывод символов во 2-й строке
                        ;с 8-й позиции (=0x80 + 0x47)
    call SchreibeDispKommando ;Передать команду для изменения адреса
                        ;памяти ЖКИ, причем первые 7 позиций
                        ;2-й строки должны остаться не пустыми
    movlw A'W'         ;Символы слова "World!"
    call SchreibeDispDaten ;последовательно друг за другом передать
    movlw A'o'         ;для отображения на индикатор
    call SchreibeDispDaten
    movlw A'r'         ;
    call SchreibeDispDaten
    movlw A'l'         ;
    call SchreibeDispDaten
    movlw A'd'         ;
    call SchreibeDispDaten
    movlw A'!'         ;Загрузить символ "!" в регистр W
    call SchreibeDispDaten ;Выполнить команду вывода последнего
                        ;символа во 2-й строке
    goto main          ;Повторить вывод текста снова и снова

```

В начале программы 4 сигнальные линии управления индикатором устанавливаются на определенный логический уровень для начала инициализации.

Затем, по вызову макрокоманды `_INIT_DISPLAY`, индикатор инициализируется. При этом делаются все необходимые установки и содержимое памяти индикатора очищается. При очистке памяти в каждую ячейку памяти пишется пробел. Затем курсор устанавливается на первую позицию в первой строке.

В основном цикле отдельные символы будут переданы на индикатор по очереди. Сначала указывается начальный адрес ячейки оперативной памяти индикатора (DDRAM) для вывода первого символа, а после этот адрес будет автоматически увеличиваться на 1 при выводе очередного символа. Для установки адреса нужно в индикатор передать команду Set DDRAM Address, код которой будет складываться из маски, указывающей на команду установки адреса ячейки памяти, и собственно самого адреса ячейки (рис. 9.8).

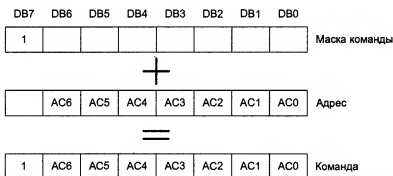


Рис. 9.8. Генерация кода команды для установки адреса оперативной памяти индикатора

После этого команда передается в контроллер индикатора.

Пример:

Если символ должен выводиться в первой строке в позиции 12, то должна послаться следующая команда:

$$\text{Код команды} = \text{Маска команды} + \text{Адрес} = 0x80 + 0x0C = 0x8C$$

Чтобы отобразить символ во второй строке в позиции 1, должна передаваться команда `0xC0`:

$$\text{Код команды} = \text{Маска команды} + \text{Адрес} = 0x80 + 0x40 = 0xC0$$

На примере вывода текста "Hello World!" можно заметить, что символы передаются как символы ASCII-кода. Подобным образом выполняется вывод, но только стандартных символов. Разумеется, так задавать специальные символы, такие, например, как Ω , при их выводе нельзя. В таком случае нужно найти символ в стандартной таблице символов контроллера индикатора ST7036, а затем выбрать соответствующий код, необходимый для отображения этого символа. Таким образом, для вывода символа единицы измерения "Ом", т. е. символа Ω , нужно использовать код `0x1E`.

10. Отображение на индикаторе аналогового напряжения

В примере вывода текста "Hello World!" представлен программный код для отображения на жидкокристаллический индикатор статического текста. Далее на примере измерения напряжения будет показано, как можно представлять на индикаторе динамические данные. С первого взгляда это кажется простым, но и здесь, разумеется, имеется несколько проблем, которые необходимо решать. Сначала аналоговое напряжение должно быть преобразовано в цифровое значение. Однако полученное значение выходного цифрового 10-битного кода не может быть непосредственно передано для отображения на индикаторе. Это значение, прежде всего, должно однозначно интерпретироваться как напряжение. Поэтому если определен цифровой код для входного измеряемого напряжения, то его нужно предварительно закодировать в ASCII-код для дальнейшего отображения на индикаторе. Чтобы все это реализовать, сначала будут рассмотрены две подпрограммы, которые выполняют соответствующие преобразования. Однако перед этим требуется немного математики, чтобы лучше понимать выполняемые вычисления.

10.1. Вычисление напряжения

Аналого-цифровой преобразователь (АЦП) выдает на выходе преобразованное двоичное 10-битное значение между 0 и 1023. В дальнейшем будем исходить из того, что для преобразования в качестве опорного напряжения используется питающее напряжение равное 5 В. Таким образом, значение 1023 соответствует напряжению 5 В. Формула для вычисления аналогового напряжения выглядит так:

$$U_{\text{ex}} = \frac{N_{\text{per}} + 0,5}{1024} \cdot U_{\text{REF}} = \frac{N_{\text{per}} + 0,5}{1024} \cdot 5 \text{ В} \quad (10.1)$$

Уже здесь появляется первая трудность. Поскольку микроконтроллер "внутри" оперирует только с целочисленными значениями, то сложение числа 0,5 и значения выходного цифрового кода регистрации $N_{\text{рег}}$ не так-то просто выполнить. Поскольку опорное напряжение получается из напряжения питания и не очень точно, то можно без особых проблем сделать маленькое упрощение в формуле, которое значительно упростит последующее вычисление. Если в формуле (10.1) опустить сложение с 0,5, то возникающей в результате этого погрешностью равной примерно 2,4 мВ в большинстве случаев можно пренебречь. Если же требуются более точные значения, то нужно предусматривать подключение внешнего АЦП, который может выполнять более точные измерения. Следовательно, с учетом названного ранее упрощения формула для вычислений будет выглядеть следующим образом:

$$U_{\text{вх}} = \frac{N_{\text{рег}}}{1024} \cdot 5 \text{ В} \quad (10.2)$$

Следующая проблема появляется, когда необходимо рассчитать аналоговое напряжение. PIC-контроллер может оперировать только с целочисленными значениями, а в этой формуле результат — это рациональное число с разрядами после запятой. Поскольку результат может определяться с точностью примерно 5 мВ, то для отображения результатов будет достаточно трех разрядов после запятой. Для дальнейшего упрощения вычислений отказываются от третьего разряда после запятой, т. е. при отображении результата с двумя разрядами после запятой погрешность будет примерно 10 мВ. Чтобы обойти эту проблему с разрядами после запятой, рассчитывают 100-кратное напряжение, а затем соответствующим образом устанавливают запятую. Умножение на 100 приводит к следующей формуле:

$$U_{\text{вх}} \cdot 100 = \frac{N_{\text{рег}}}{1024} \cdot 100 \cdot 5 \text{ В} \quad (10.3)$$

Чтобы оптимизировать последующее вычисление, имеет смысл еще немного преобразовать формулу и "подогнать" ее для вычисления с помощью микроконтроллера. Поскольку у PIC-микроконтроллера нет команд для умножения и деления, поэтому эти математические операции должны осуществляться действиями арифметики — сложением, вычитанием и операциями сдвига. Так, операцию умножения на 2 можно заменить выполнением одного сдвига битов влево, а деления на 2 — сдвигом вправо. Поэтому формулу (10.3) нужно оптимизировать таким образом, чтобы можно было обходиться этими 4 командами (сложения, вычитания, сдвига влево и сдвига вправо). Для этого можно предпринимать следующие преобразования:

$$\begin{aligned}
 U_{\text{вх}} \cdot 100 &= \frac{N_{\text{пер}}}{1024} \cdot 100 \cdot 5 \text{ В} = N_{\text{пер}} \cdot \frac{500}{1024} = N_{\text{пер}} \cdot \frac{400 + 100}{1024} \\
 U_{\text{вх}} \cdot 100 &= N_{\text{пер}} \cdot \left(\frac{400 + 100}{1024} \right) = N_{\text{пер}} \cdot \left(\frac{100}{256} + \frac{100}{1024} \right) \\
 U_{\text{вх}} \cdot 100 &= N_{\text{пер}} \cdot \left(\frac{25}{64} + \frac{25}{256} \right) = N_{\text{пер}} \cdot \left(\frac{25}{64} + \frac{25}{64 \cdot 2 \cdot 2} \right) \quad (10.4)
 \end{aligned}$$

Из преобразования видно, что выходной цифровой код преобразователя нужно умножать на 25 и делить затем на 64. Деление на 64 может реализоваться неоднократными сдвигами вправо, т. к. речь идет о степени 2. Чтобы лучше можно было понять последующую подпрограмму, выполним определение следующих регистров:

ADC_L	EQU	0x30	; Копия значения АЦП
ADC_H	EQU	0x31	
ADC_Lx25	EQU	0x32	; Значение АЦП умноженное на 25
ADC_Hx25	EQU	0x33	
ADC_Lx25_64	EQU	0x34	; Значение АЦП умноженное на 25/64
ADC_Hx25_64	EQU	0x35	
ADC_Lx25_64	EQU	0x36	; Значение АЦП умноженное на 25/256
ADC_Hx25_64	EQU	0x37	
Ux100_L	EQU	0x38	; Значение напряжения умноженное на 100
Ux100_H	EQU	0x39	

Чтобы реализовать умножение на 25, можно разделить это действие на следующие операции:

$$25 = (2 + 1) \cdot 2 \cdot 2 + 1$$

Используя приведенные упрощения, можно написать подпрограмму аналого-цифрового преобразования `AD_konvertieren`.

10.2. Подпрограмма "AD_konvertieren"

```

AD_konvertieren
    _BANK_0
    movf ADRESH, W      ;Получить старший байт значения АЦП в регистре W
    ;movlw 0x01         ;Здесь можно перезаписать значение АЦП
    ;                    ;тестовым значением 0x01
    movwf ADC_H          ;Старший байт значения АЦП скопировать
    ;                    ;в регистр ADC_H
    movwf ADC_Hx25       ;Старший байт значения АЦП еще раз скопировать
    ;                    ;в регистр ADC_Hx25

```

```

_BANK_1
movf ADRESL, W      ;Получить младший байт значения АЦП в регистре W
_BANK_0
;movlw 0xE0          ;Здесь можно перезаписать значение АЦП
;                    ;тестовым значением 0xE0
movwf ADC_L          ;Младший байт значения АЦП скопировать
;                    ;в регистр ADC_L
movwf ADC_Lx25        ;Младший байт значения АЦП еще раз скопировать
;                    ;в регистр ADC_Lx25

;Умножение на 25
bcf STATUS, C        ;Сбросить флаг переноса перед операцией сдвига
rlf ADC_Lx25, F        ;Умножить на 2 с помощью сдвига влево
rlf ADC_Hx25, F        ;содержимого регистров ADC_Lx25 и ADC_Hx25
movf ADC_L, W          ;Сложение значений
addwf ADC_Lx25, F      ;(соответствует в общей сложности умножению на 3)
btfsc STATUS, C
incf ADC_Hx25, F
movf ADC_H, W
addwf ADC_Hx25, F
bcf STATUS, C          ;Сбросить флаг переноса перед операцией сдвига
rlf ADC_Lx25, F        ;Выполнить операцию сдвига битов влево, что
;                    ;в общей сложности соответствует умножению на 6
bcf STATUS, C          ;Сбросить флаг переноса перед операцией сдвига
rlf ADC_Lx25, F        ;Выполнить операцию сдвига битов влево, что
;                    ;в общей сложности соответствует умножению на 12
bcf STATUS, C          ;Сбросить флаг переноса перед операцией сдвига
rlf ADC_Lx25, F        ;Выполнить операцию сдвига битов влево, что
;                    ;в общей сложности соответствует умножению на 24
movf ADC_L, W          ;Добавить еще одно значение, что
;                    ;в общей сложности соответствует умножению 25
addwf ADC_Lx25, F      ;Поместить младший байт результата умножения
;                    ;на 25 в регистр ADC_Lx25

btfsc STATUS, C
incf ADC_Hx25, F
movf ADC_H, W
addwf ADC_Hx25, F      ;Поместить старший байт результата умножения
;                    ;на 25 в регистр в ADC_Hx25
movf ADC_Hx25, W        ;Скопировать содержимое регистра ADC_Hx25
movwf ADC_Hx25_64       ;в регистр ADC_Hx25_64
movf ADC_Lx25, W        ;Скопировать содержимое регистра ADC_Lx25
movwf ADC_Lx25_64       ;в регистр ADC_Lx25_64
;Значение АЦП 25 раз делится на 64
;(выполнение операции сдвига вправо 6 раз)
bcf STATUS, C          ;Сбросить флаг переноса перед операцией сдвига

```

```

rrf ADC_Hx25_64, F ;Выполнить операцию сдвига вправо,
rrf ADC_Lx25_64, F ;которая соответствует делению на 2
bcf STATUS, C ;Сбросить флаг переноса перед операцией сдвига
rrf ADC_Hx25_64, F ;Выполнить операцию сдвига вправо, что в этот
rrf ADC_Lx25_64, F ;момент уже соответствует делению на 4
bcf STATUS, C ;Сбросить флаг переноса перед операцией сдвига
rrf ADC_Hx25_64, F ;Выполнить операцию сдвига вправо, что в этот
rrf ADC_Lx25_64, F ;момент соответствует делению на 8
bcf STATUS, C ;Сбросить флаг переноса перед операцией сдвига
rrf ADC_Hx25_64, F ;Выполнить операцию сдвига вправо, что в этот
rrf ADC_Lx25_64, F ;момент соответствует делению на 16
bcf STATUS, C ;Сбросить флаг переноса перед операцией сдвига
rrf ADC_Hx25_64, F ;Выполнить операцию сдвига вправо, что в этот
rrf ADC_Lx25_64, F ;момент соответствует делению на 32
bcf STATUS, C ;Сбросить флаг переноса перед операцией сдвига
rrf ADC_Hx25_64, F ;Выполнить операцию сдвига вправо, что в этот
rrf ADC_Lx25_64, F ;момент соответствует делению на 64
movf ADC_Hx25_64, W ;Скопировать содержимое регистра ADC_Hx25_64
movwf ADC_Hx25_256 ;в регистр ADC_Hx25_256
movf ADC_Lx25_64, W ;Скопировать содержимое регистра ADC_Lx25_64
movwf ADC_Lx25_256 ;в регистр ADC_Lx25_256
;Значение АЦП 25 раз делится на 256
;(результат деления на 64, выполняя
; 2 сдвига вправо, еще делят на 4)
bcf STATUS, C ;Сбросить флаг переноса перед операцией сдвига
rrf ADC_Hx25_256, F ;Выполнить операцию сдвига вправо, что
rrf ADC_Lx25_256, F ;соответствует делению на 128
bcf STATUS, C ;Сбросить флаг переноса перед операцией сдвига
rrf ADC_Hx25_256, F ;Выполнить операцию сдвига вправо, что
rrf ADC_Lx25_256, F ;соответствует делению на 256
;Сложение значений ADCx25/64 и ADCx25/256
movf ADC_Lx25_64, W
addwf ADC_Lx25_256, W
movwf Ux100_L ;Значение напряжения умноженное на 100 (мл. байт)
btfsc STATUS, C
incf ADC_Hx25_256, F
movf ADC_Hx25_64, W
addwf ADC_Hx25_256, W
movwf Ux100_H ;Значение напряжения умноженное на 100 (ст. байт)
return

```

После выполнения подпрограммы значение измеренного напряжения, умноженное на 100, будет храниться в обоих регистрах Ux100_H и Ux100_L и может

использоваться в дальнейшем. При выполнении операций сдвига нужно обращать внимание на то, что флаг переноса перед операцией обязательно должен быть сброшен (очищен), поскольку иначе значение флага после операции сдвига может исказить результат, заняв место младшего бита регистра. В начале текста подпрограммы находятся две закомментированных строки. Если нужно имитировать программу и проверить вычисление, то можно разрешить выполнение этих обеих команд и тем самым перезаписать результат, полученный АЦП. Таким образом, можно задать любое значение для АЦП и просматривать выполнение вычислений в подпрограмме. Однако эти строки должны быть закомментированы или же удалены перед загрузкой программы на микроконтроллер, поскольку иначе на индикаторе всегда будет отображаться одно и то же тестовое значение.

10.3. Преобразование двоичного числа в десятичное число

После выполнения подпрограммы `AD_konvertieren` будет получено рассчитанное двоичное значение напряжения. Разумеется, оно еще не может быть отображено на жидкокристаллическом индикаторе. Поэтому необходимо еще дополнительное преобразование. Значение напряжения — это число между 0 и 500 и оно имеет три десятичных разряда. Чтобы определить разряд сотен, нужно разделить значение напряжения на 100 и таким образом получить место для запятой. К сожалению, выполнить деление не так-то просто. Поэтому нужно обратиться к операции вычитания и вычитать из значения напряжения 100 так долго, пока результат не станет отрицательным. Количество возможных вычитаний тогда укажет на число сотен. Так же поступают с последним положительным получившимся остатком для определения числа десятков. Последний получившийся остаток, который получается после неоднократного вычитания 10, указывает на значение числа единиц.

Пример:

Определение числа сотен.

$321 - 100 = 221$ (1-е успешное вычитание)

$221 - 100 = 121$ (2-е успешное вычитание)

$121 - 100 = 21$ (3-е успешное вычитание)

$21 - 100 = -79$ (Отрицательное значение. Вычитать больше не нужно).

Число 100 вычиталось три раза без возникновения отрицательного результата. Поэтому значение числа сотен будет равно 3.

Определение числа десятков:

$21 - 10 = 11$ (1-е успешное вычитание)

$11 - 10 = 1$ (2-е успешное вычитание)

$1 - 10 = -9$ (Отрицательное значение. Вычитать больше не нужно).

Число 10 дважды успешно вычиталось, таким образом значение числа десятков будет равно 2.

Число единиц — это остаток, который остается после последнего успешного вычитания 10, и, как видно из примера, оно равно 1.

Описанный метод вычисления и применяется в следующем примере подпрограммы B2D.

Пример:

```

B2D
    _BANK_0
    movf Ux100_H, W      ;Скопировать значение напряжения для вычислений
    movwf CALC_H_1
    movf Ux100_L, W
    movwf CALC_L_1
    clrf COUNTER         ;Счетчик, который считает количество успешных
                           ;вычитаний
subtrahieren_H           ;Вычитание сотни
    movlw d'100'         ;Число 100 для вычитаний
    movwf CALC_L_2        ;поместить в регистр CALC_L_2
    clrf CALC_H_2         ;Очистить регистр CALC_H_2, поскольку для
                           ;вычислений он не требуется
    call Sub16            ;Выполнить 16-битное вычитание 100 из значения
                           ;напряжения в регистре Ux100_H/L
    btfsc STATUS, C       ;Проверить, результат положительный или
                           ;отрицательный
    goto ergebnis_Pos_H  ;Если результат положительный, то выполнить
                           ;вычитание 100 еще раз
    goto ergebnis_Neg_H  ;Если результат отрицательный, то число сотен
                           ;теперь известно
ergebnis_Pos_H
    incf COUNTER, F       ;Счетчик числа сотен увеличить и еще
    goto subtrahieren_H   ;раз перейти к процедуре вычитания 100
ergebnis_Neg_H
    movlw d'100'          ;Поскольку число 100 вычли один лишний раз,
                           ;то нужно его снова добавить
    movwf CALC_L_2
    clrf CALC_H_2

```

```

call Add16           ;Добавление 100
movf COUNTER, W      ;В регистре COUNTER хранится
                     ;число сотен в значении измеренного напряжения
movwf DISP_HUNDERTER ;Скопировать содержимое регистра COUNTER в
                     ;регистр DISP_HUNDERTER
clrf COUNTER         ;Очистить счетчик COUNTER
                     ;для последующих вычислений
subtrahieren_2       ;Вычитание десятка
movlw d'10'          ;Число 10 для вычитаний
movwf CALC_L_2       ;поместить в регистр CALC_L_2
clrf CALC_H_2        ;Очистить регистр CALC_H_2, поскольку для
                     ;вычислений он не требуется
call Sub16           ;Выполнить 16-битное вычитание 10 из
                     ;оставшегося значения напряжения
btfsc STATUS, C      ;Проверить, результат положительный или
                     ;отрицательный
goto ergebnis_Pos_2 ;Если результат положительный, то выполнить
                     ;вычитание 10 еще раз
goto ergebnis_Neg_2 ;Если результат отрицательный, то число десятков
                     ;теперь известно
ergebnis_Pos_2
incf COUNTER, F      ;Счетчик числа десятков увеличить и еще
goto subtrahieren_2 ;раз перейти к процедуре вычитания 10
ergebnis_Neg_2
movlw d'10'          ;Поскольку число 10 вычли один лишний раз,
                     ;то нужно его снова добавить
movwf CALC_L_2
clrf CALC_H_2
call Add16           ;Добавление 10
movf COUNTER, W      ;В регистре COUNTER хранится
                     ;число десятков в значении измеренного напряжения
movwf DISP_ZEHNER    ;Скопировать содержимое регистра COUNTER
                     ;в регистр DISP_ZEHNER
movf CALC_L_1, W     ;Получение числа единиц из
                     ;содержимого регистра CALC_L_1
movwf DISP_EINER     ;Копирование этого числа в регистр DISP_EINER
return

```

В начале подпрограммы значение измеренного напряжения, которое умножалось на 100, загружается в регистры, предназначенные для выполнения операции 16-битного вычитания. Поскольку в данном случае в вычислениях могут использоваться значения большие, чем 8 битов, то для вычитания используется подпрограмма именно 16-битного вычитания Sub16 (см. главу 8). После вычитания 100 проверяется, отрицателен ли результат или же положи-

телен. В случае положительного результата будет переход на метку `ergebnis_Pos_H` для продолжения вычитания, а счетчик будет увеличен на единицу. Если результат отрицателен, то будет переход на метку `ergebnis_Neg_H`. Поскольку в этом случае из значения вычли сотню лишний раз, то снова должно быть добавлено число 100, чтобы было возможно следующее вычисление. После выполнения операции сложения значение счетчика (`COUNTER`) копируется в регистр `DISP_HUNDERTER`. Вслед за этим аналогичным образом рассчитывается число десятков в измеренном напряжении. Однако перед этим счетчик `COUNTER` должен быть обнулен. После вычисления значений десятков число копируется в регистр `DISP_ZEHNER`. Полученный остаток будет искомым числом единиц, которое также сохраняется в отведенном для этого регистре `DISP_EINER`.

10.4. Основная программа

Благодаря подготовительной работе, выполненной в обеих подпрограммах, основная программа относительно коротка и наглядна. В ней должны инициализироваться жидкокристаллический индикатор и АЦП, а затем в основном цикле текущее измеренное значение напряжения должно выводиться на индикатор.

```
start                                     ;Начало программы
;Initialisierungen
_BANK_0
clrf PORTA                               ;Очистить все регистры портов
clrf PORTB
clrf PORTC
_BANK_1
movlw b'10001110'                       ;Определить вывод AN0, как аналоговый вход
movwf ADCON1                             ;Результат выравнивать по правому краю
movlw b'11110011'                       ;Определить выводы RA3 и RA2 порта A, как
movwf TRISA                              ;цифровые выходы, а остальные – входы
movlw b'11000000'                       ;Определить выводы RB7 и RB6 порта B, как
movwf TRISB                              ;цифровые входы, а остальные – выходы
movlw b'11111111'                       ;Все выводы порта C – цифровые входы
movwf TRISC
_BANK_0
movlw b'01000000'                       ;Инициализировать АЦП, канал 0 (AN0),
;Fosc/8 (макс. тактовая частота 5 МГц)

movwf ADCON0
bsf DISP_CSB                             ;Подать на индикатор сигнал #CSB высокого
;логического уровня -> ЖКИ пассивен
```

bcf DISP_CLK	;Установить сигнал CLK низкого уровня
bcf DISP_SI	;Установить сигнал SI низкого уровня
bcf DISP_RS	;На вывод RS передать низкий уровень,
	;соответствующий передаче команд
_INIT_DISPLAY	;После инициализации содержимое
	;индикатора очистить, или иначе
	;заполнить его пробелом
main	;Начало основного цикла
movlw 0x80	;Задать вывод символов в 1-й строке
	;с 1-й позиции (=0x80 + 0x00)
call SchreibeDispKommando	;Передать команду для выполнения
	;установки адреса памяти индикатора
movlw A'U'	;Загрузить символ "U" в регистр W для
	;вывода в 1-й строке с 1-й позиции
call SchreibeDispDaten	;Выполнить передачу символа
movlw A'='	;Загрузить символ "=" в регистр W
call SchreibeDispDaten	;Выполнить передачу символа с
	;автоматическим увеличением адреса для
	;вывода символа в 1-ю строку во 2-ю позицию
lese_AN0	;Считывание напряжения с вывода AN0
_BANK_1	
clrf ADRESL	;Очистить младший байт регистра для
	;хранения результата АЦП
_BANK_0	
clrf ADRESH	;Очистить старший байт регистра для
	;хранения результата АЦП
bsf ADCON0, ADON	;Включить модуль АЦП
bsf ADCON0, GO_DONE	;Запустить преобразование АЦП
btfsc ADCON0, GO_DONE	
goto \$-1	;Ожидать, пока аппаратно не будет
	;возвращен бит GO_DONE
nop	;Задержка для завершения преобразования АЦП
call AD_konvertieren	;Преобразование результата АЦП в
	;значение напряжения
call B2D	;Преобразование двоичного значения в три
	;десятичных разряда измеренного напряжения
movlw 0x82	;Задать 3-ю позицию в 1-й строке ЖКИ
call SchreibeDispKommando	
movlw 0x30	;Преобразование числа сотен
	;в код ASCII для вывода
addwf DISP_HUNDERTER, W	;Код ASCII = числовое значение + 0x30
call SchreibeDispDaten	
movlw A'.'	


```

call SchreibeDispDaten      ;Отображение десятичной точки
movlw 0x30                  ;Преобразование числа десятков в код ASCII
addwf DISP_ZEHNER, W
call SchreibeDispDaten      ;Отображение крайнего левого разряда
                             ;после запятой
movlw 0x30                  ;Преобразование числа единиц в код ASCII
addwf DISP_EINER, W
call SchreibeDispDaten      ;Отображение крайнего второго разряда
                             ;после запятой
movlw A'V'
call SchreibeDispDaten      ;Отображение числа единиц напряжения
goto main                  ;Перезапуск измерения

```

Во время инициализации микроконтроллера вывод AN0 определяется в качестве аналогового входа, а с помощью макрокоманды `_INIT_DISPLAY` инициализируется индикатор. В основном цикле `main` на жидкокристаллический индикатор сначала передаются два символа "U =", которые отображаются, начиная с первой позиции первой строки. После этого может быть запущена процедура измерения напряжения, которую будет выполнять АЦП. Для этого сначала очищаются предыдущие значения в регистрах `ADRESL` и `ADRESH`, а затем включается модуль АЦП. В следующем цикле программа ожидает завершения преобразования и получения измеренного значения в регистрах. Далее вызываются две подпрограммы `AD_konvertieren` и `B2D` для преобразования результата АЦП в значение напряжения и последующего преобразования его двоичного значения в десятичное. После завершения подпрограмм в регистры `DISP_HUNDERTER`, `DISP_ZEHNER` и `DISP_EINER` будет записано три десятичных разряда. Если передать эти значения непосредственно на индикатор, то при этом никакого отображения чисел не было бы, а выводились только лишь какие-нибудь случайные специальные символы, поскольку для отображения символов на индикаторе должна использоваться определенная таблица кодов. Если внимательно просмотреть на эту таблицу, то можно заметить, что цифра "0" имеет код 0x30, "1" — код 0x31, "2" — код 0x32 и т. д. вплоть до цифры "9", которая имеет код 0x39. Поэтому для верного отображения десятичных чисел результата к полученным значениям должно добавляться шестнадцатеричное значение 0x30, что и выполняется перед передачей данных на индикатор.

Если, например, содержимое регистра `DISP_HUNDERTER`, предназначенного для хранения разряда сотен, имеет значение 3, то на индикатор будет передан код 0x33 для отображения именно символа "3". После передачи всех разрядов числа измеренного напряжения, а также десятичной точки отображение завершается выводом символа для единицы измерения напряжения "V" (вольт). Затем аналоговое значение по команде `goto main` в бесконечном цикле снова

и снова считывается и выводится на индикатор. Ранее переданные на индикатор символы перезаписываются при каждом новом измерении. Если же потребуется немного замедлить отображение значений на индикаторе, то можно перед командой `goto main` добавить код для задания небольшого времени ожидания, примерно от 100 до 300 мс.

Этот пример очень хорошо может быть воспроизведен на демонстрационной плате. При изменении входного напряжения соответствующим образом будет изменяться и отображение на индикаторе. Для контролирования напряжения можно использовать дополнительный вольтметр, с помощью которого определяют, насколько точно микроконтроллер выполняет измерение входного напряжения.

11. Измерение мощности и сопротивления

В предыдущем примере было представлено измерение напряжения, подаваемого на аналоговый вход, а также отображение оцифрованного значения напряжения на жидкокристаллическом индикаторе. Однако в электротехнике часто требуется отображение не только напряжения, но и потребляемого тока. В этой главе приведен следующий пример, который показывает, как разделить напряжение и потребляемый ток, а также как из этих двух полученных значений можно рассчитать мощность ($P = U \times I$) и сопротивление ($R = U / I$).

11.1. Измерение тока

Поскольку встроенный в микроконтроллер АЦП не может измерять ток, то для его измерения нужно применить дополнительную схему для преобразования тока в напряжение. Для измерения тока в нагрузке последовательно с ней включают резистор с заранее известным и гораздо меньшим, чем нагрузка, сопротивлением и затем на нем измеряют падение напряжения. На рис. 11.1 представлена принципиальная схема, на которой показано, как можно измерить ток в нагрузке. Схема представлена только для примера и условно пригодна для использования на практике, поскольку дополнительная погрешность, вносимая этим небольшим сопротивлением, будет относительно высока.

На примере этой схемы очень хорошо видно, как можно с помощью дополнительного усилителя предоставить в распоряжение микроконтроллера необходимое для измерения напряжение. В этой схеме измеренное напряжение на аналоговом входе AN1 равно 1 В соответствует протекающему в цепи току в 1 А. Таким образом, возможен очень простой пересчет измеряемого напряжения в ток. Напряжение должно измеряться на входе AN1 и при отображении значения на индикаторе вместо символа "V" (вольт), как единицы измерения напряжения, должен выводиться символ "A" (ампер), как единицы измерения тока.

При измерении надо учесть, что PIC-микроконтроллер имеет только один внутренний АЦП, который правда может подключаться к различным входам. Поскольку напряжение на входах AN0 и AN1 не может измеряться одновременно, то нужно иметь в виду, что при быстрых изменениях входного измеряемого сигнала рассчитанное значение напряжений на обоих входах может быть ошибочно. Подобное может произойти, когда после измерения напряжения на входе AN0 при последующем измерении напряжения на входе AN1 по какой-либо причине изменяется сопротивление нагрузки.

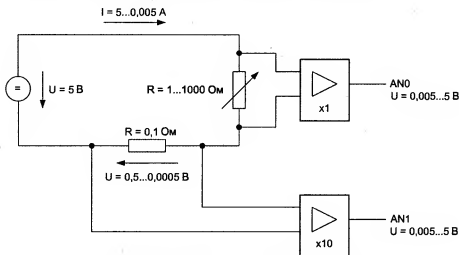


Рис. 11.1. Простейшая схема для измерения тока

11.2. Двоичное умножение

Чтобы определить мощность, после получения двух оцифрованных значений напряжения и тока они должны быть перемножены между собой. Однако для микроконтроллера это не такая простая задача, как выглядит на первый взгляд. Небольшие микроконтроллеры, в том числе и PIC16F876A, не располагают командами умножения. Поэтому операция умножения должна выполняться посредством основных арифметических функций, т. е. операций суммирования и сдвига. В самом простейшем случае умножение может выполняться даже путем многократного сложения.

Пример:

$$15 * 5 = 15 + 15 + 15 + 15 + 15 = 75$$

Этот метод вычисления имеет смысл для относительно небольших чисел, поскольку для расчета результата требуется малое количество операций сложения.

ния. Если же требуется обработка больших чисел, то этот метод вычисления не выгоден, и нужно использовать более эффективные алгоритмы. Чтобы разработать такой алгоритм, поступают как при письменном умножении десятичных чисел в столбик. Для лучшего и точного понимания принципа умножения письменно выполним небольшой пример. Для примера умножим число 147 на 86.

```

147
* 86
----
602 = 7 * 86 (Разряд единиц)
344 = 4 * 86 (Разряд десятков)
86  = 1 * 86 (Разряд сотен)
-----
12642 = 602 + (10 * 344) + (100 * 86)
=====

```

При письменном умножении процесс умножения разбивается на маленькие шаги, при которых промежуточный результат записывается в столбец со сдвигом соответствующего разряда. Конечный результат получается при последующем сложении сдвинутых значений.

Умножение двух двоичных чисел происходит таким же способом и даже еще несколько проще, т. к. в двоичных числах имеются только две возможные цифры для каждого разряда (1 и 0). Вследствие этого вычисление может существенно упроститься.

```

10010011 = 147 = 0x93
* 01010110 = 86 = 0x56
-----
01010110 = 1 * 86
01010110 = 1 * 86
00000000 = 0 * 86
00000000 = 0 * 86
01010110 = 1 * 86
00000000 = 0 * 86
00000000 = 0 * 86
01010110 = 1 * 86
-----
0011000101100010 = 12642 = 0x3162
=====

```

На примере видно, что множитель умножается либо на 1, либо на 0. При умножении на 0 результат также равен 0 и при умножении на 1 результатом будет само число. На практике сложение выполняется уже после каждого

умножения на 0 или 1, т. к. таким образом можно значительно уменьшить количество необходимых для операции умножения регистров.

Если внимательно посмотреть на пример, то можно заметить, что количество необходимых разрядов для результата равно совместному количеству битов множимого и множителя (рис. 11.2). Поэтому при умножении двух 8-битных значений для хранения результата требуется 16-битный регистр.

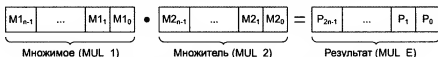


Рис. 11.2. Регистры для выполнения операции умножения

Теперь, исходя из сведений предыдущего примера, можно разработать простой алгоритм. Для блок-схемы выполнения программы (рис. 11.3) использовались обозначения, примененные на рис. 11.2. В последующей подпрограмме так же используются обозначения MUL_1, MUL_2 и MUL_E.

В следующей подпрограмме mul16 показана практическая реализация умножения содержимого двух 16-битных регистров. Результат после вычисления будет храниться в 32-битном регистре, который состоит в PIC-контроллере из четырех 8-битных регистров.

Mul16

```
clrf MUL_E_4      ;Очистка регистра для хранения предыдущего
clrf MUL_E_3      ;результата, состоящего из 4-х
clrf MUL_E_2      ;8-разрядных регистров
clrf MUL_E_1
bsf MUL_E_2, d'7'  ;Установить бит 7 в регистре результата MUL_E_2
                  ;для определения выполнения 16-ти операций сдвига
```

mul16_loop

```
rrf MUL_1_H, F    ;Циклический сдвиг вправо содержимого рег. MUL_1
rrf MUL_1_L, F    ;Если МЗР (LSB) = 1, то устанавливается бит C
                  ;(переноса) и должно быть выполнено сложение
btfss STATUS, C   ;Если же бит C = 0, выполняется умножение на 0
goto mul16_rotate ;и ничего не складывается
movf MUL_2_L, W   ;Начало 16-битного сложения
addwf MUL_E_3, F
movf MUL_2_H, W
btfsc STATUS, C   ;Проверить, установлен ли бит переноса (C = 1)
incfsz MUL_2_H, W ;Если да, то добавить 1 к старшему байту
                  ;множителя
addwf MUL_E_4, F  ;Если нет, то 1 к старшему байту множителя
                  ;не добавлять
```

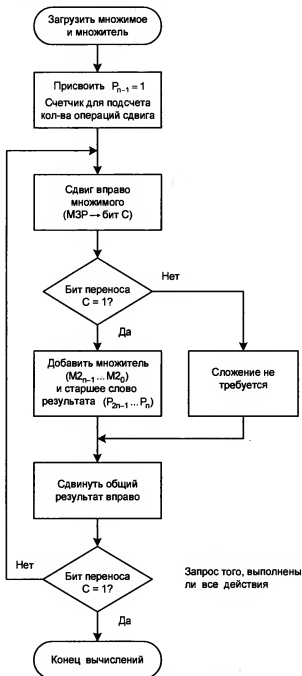


Рис. 11.3. Блок-схема выполнения программы умножения

```

mul16_rotate
    rrf MUL_E_4, F      ;Выполнить сдвиг вправо содержимого 32-битного
    rrf MUL_E_3, F      ;регистра временного хранения результата
    rrf MUL_E_2, F
    rrf MUL_E_1, F
    btfs STATUS, C      ;Если бит 7 из регистра MUL_E_2 поступил в бит C,
                        ;то в 32-битном регистре будет итоговый результат,
    goto mul16_loop     ;поскольку 16 операций сдвига выполнены и
                        ;вычисление результата умножения завершено
    return              ;Возврат из подпрограммы

```

Перед вызовом подпрограммы с помощью команды `call Mul16` умножаемые значения должны загружаться в регистры `MUL_1_H/MUL_1_L` и `MUL_2_H/MUL_1_L`. После возврата из подпрограммы результат находится в четырех 8-разрядных регистрах `MUL_E_4—MUL_E_1`.

11.3. Двоичное деление

Чтобы рассчитать значение сопротивления из двух полученных оцифрованных значений, нужно разделить оцифрованное значение напряжения на значение тока $R = U / I$. Поскольку в микроконтроллере отсутствует команда деления, так же как и команда умножения, то поэтому для реализации операции деления нужно воспользоваться имеющимися арифметическими командами. Операция деления требует очень большого объема вычислений, особенно если еще должны определяться разряды после запятой. При вычислении разрядов после запятой нужно отменять вычисление для рациональных чисел (чисел, представляемых обыкновенной дробью, где числитель — целое число, а знаменатель — натуральное число), поскольку расчет может быть бесконечным (например, $8 : 3 = 2,666\dots$). На практике целесообразное количество разрядов после запятой определяется, как правило, точностью измерения. Самое маленькое напряжение, которое может воспринимать АЦП микроконтроллера, составляет примерно 5 мВ. Поэтому нет большого смысла вычислять более 3 разрядов после запятой.

Чтобы выполнить деление с помощью микроконтроллера, имеются, как и для умножения, несколько возможностей. С одной стороны, можно многократно вычитать значение знаменателя из числителя до тех пор, пока полученное значение не будет меньше 0. Количество вычитаний сохраняется в счетчике. К сожалению, этот очень простой метод может потребовать очень много времени, если знаменатель имеет маленькое значение, а числитель очень большое. Чтобы написать подпрограмму, которая будет выполнять операцию деления, обращаются к алгоритму, который упрощает деление. Поскольку при делении оба числа могут быть не целочисленными и не кратными друг другу,

то в результате возникает остаток, и поэтому нужно предусматривать также регистр для хранения остатка. Для последующего далее алгоритма выбирается приведенное на рис. 11.4 распределение регистров.

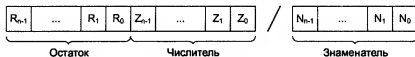


Рис. 11.4. Регистр для деления

Перед выполнением деления в заранее отведенные для этого регистры загружают нужные значения числителя и знаменателя. В конце деления результат будет в числителе, а остаток сохраняется в соразмерном соответствующем регистре. С помощью этого метода экономится регистр для хранения результата, а также в связи с этим дополнительные затраты на копирование для сдвига значений в регистре результата. На рис. 11.5 представлена блок-схема алгоритма для выполнения операции деления. Однако кроме регистров числителя, знаменателя и остатка требуется дополнительный регистр для счетчика (Counter), который будет подсчитывать количество операций. Вначале этот счетчик загружается количеством делящихся битов.

С помощью этого алгоритма результат рассчитывается, как и при умножении, путем последовательных операций сдвига и вычитания. Для пояснения процесса вычисления приведен пример деления двух 8-битных значений (рис. 11.6). В данном примере число 126 (0x7E) делится на 3 (0x03). После вычисления в итоге получается результат равный 42 (0x2A) с остатком 0 (0x00).

В примере очень хорошо видно, как сначала числитель сдвигается влево, при этом старший значащий разряд СЗР (MSB) числителя переходит в регистр остатка и после этого из остатка вычитается знаменатель. Если при вычитании будет получен положительный результат, то младший значащий разряд (МЗР) числителя Z_0 устанавливается в 1. Если же результат был отрицательным (бит переноса $C = 0$), то для отмены выполненного вычитания снова знаменатель добавляется к остатку. В принципе результат операции деления можно считать по значениям бита переноса C (Carry).

Согласно схеме, представленной для примера на рис. 11.1, значение сопротивления должно быть рассчитано с 3 разрядами после запятой. Чтобы это гарантировать, значение напряжения умножается на коэффициент 1000 и запятая устанавливается после вычисления в соответствующем разряде. Так как оцифрованное значение напряжения может составлять максимум 1023, то 1000-кратное значение напряжения будет равно 1023000. Для отображения этого значения требуются минимум 20 битов. Поэтому следующая под-



Рис. 11.5. Блок-схема алгоритма для выполнения деления

Счетчик	Бит С	Остаток	Числитель	Знаменатель	Примечание
n = 8	0	0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 0	1 1 1 1 1 1 1 1 1 0	0 0 0 0 0 0 0 1 1	Исходное состояние
		0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 0 1		0 0 0 0 0 0 0 1 1	Сдвиг влево Остаток – Знаменатель Результат
n = 7		0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0			Результат отрицательный Остаток + Знаменатель
	0	0 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 0		0 0 0 0 0 0 0 1 1	Сдвиг влево Остаток – Знаменатель Результат
n = 6		0 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0			Результат отрицательный Остаток + Знаменатель
	1	0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0		0 0 0 0 0 0 0 1 1	Сдвиг влево Остаток – Знаменатель Результат
n = 5		0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 1			Результат положительный МЗР числителя = 1
	0	0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 1 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 0		0 0 0 0 0 0 0 1 1	Сдвиг влево Остаток – Знаменатель Результат
n = 4		0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 1 0			Результат отрицательный Остаток + Знаменатель
	1	0 0 0 0 0 0 0 1 1 1 1 0 0 0 1 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0		0 0 0 0 0 0 0 1 1	Сдвиг влево Остаток – Знаменатель Результат
n = 3		0 0 0 0 0 0 0 0 0 1 1 0 0 0 1 0 1 1			Результат положительный МЗР числителя = 1
	0	0 0 0 0 0 0 0 0 1 1 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 0		0 0 0 0 0 0 0 1 1	Сдвиг влево Остаток – Знаменатель Результат
n = 2		0 0 0 0 0 0 0 0 1 1 0 0 0 0 1 0 1 0			Результат отрицательный Остаток + Знаменатель
	1	0 0 0 0 0 0 0 1 1 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0		0 0 0 0 0 0 0 1 1	Сдвиг влево Остаток – Знаменатель Результат
n = 1		0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1			Результат положительный МЗР числителя = 1
	0	0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 0 1		0 0 0 0 0 0 0 1 1	Сдвиг влево Остаток – Знаменатель Результат
n = 0		0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0			Результат отрицательный Остаток + Знаменатель
Результат =		0 0 1 0 1 0 1 0			
Остаток =		0 0 0 0 0 0 0 0			

Рис. 11.6. Пример деления

программа для деления выполняет деление 24-битного значения числителя на 16-битное значение знаменателя. Перед вызовом подпрограммы требуется обеспечить предварительную загрузку регистров ZAEHLER_L (младший байт), ZAEHLER_M (средний байт) и ZAEHLER_H (старший байт), предназначенных для хранения значения числителя. Значение знаменателя должно загружаться в два регистра: NENNER_L (младший байт) и NENNER_H (старший байт).

Div16

```
    clrf REST_H           ;Очистить содержимое регистра, предназначенного
    clrf REST_L           ;для хранения остатка
    movlw d'24'           ;Загрузить 24 в регистр счетчика DIV_COUNTER,
    movwf DIV_COUNTER     ;поскольку числитель имеет 24 бита
```

div16_loop

```
    rlf ZAEHLER_L, F      ;Выполнить циклический сдвиг влево через бит C
    rlf ZAEHLER_M, F      ;содержимого регистра числителя
    rlf ZAEHLER_H, F
    rlf REST_L, F         ;Выполнить циклический сдвиг влево через бит C
    rlf REST_H, F         ;содержимое регистра для хранения остатка
    rlf DIV_COUNTER, F    ;Бит переноса C временно сохраняется в регистре
                           ;DIV_COUNTER, вследствие этого
                           ;можно сэкономить регистр
```

```
    movf NENNER_L, W
    subwf REST_L, F       ;Вычесть младший байт знаменателя Nenner_L из
                           ;остатка Rest_L (REST_L - NENNER_L => REST_L)
```

```
    movf NENNER_H, W
    btfss STATUS, C
    incfsz NENNER_H, W
    subwf REST_H, W       ;Вычесть старший байт знаменателя Nenner_H из
                           ;остатка Rest_H (REST_H - NENNER_H => W)
```

```
    btfsc STATUS, C       ;Проверить, результат положительный?
    bsf DIV_COUNTER, 0    ;Если да, то установить значение сохраненного
                           ;бита переноса C в регистре DIV_COUNTER
    btfsc DIV_COUNTER, 0  ;Проверить сохраненный бит переноса C в
                           ;регистре DIV_COUNTER
```

```
    goto div16_jump      ;Если бит C = 1, перейти на метку div16_jump
    movf NENNER_L, W      ;Если бит C = 0 (отрицательный результат), то
    addwf REST_L, F       ;отменить ранее выполненное вычитание
    movf REST_H, W
```

div16_jump

```
    movwf REST_H          ;Сохранить остаток из регистра W в рег. REST_H
    bcf STATUS, C         ;Очистить бит переноса (C = 0) и
    rrf DIV_COUNTER, F    ;затем восстановить содержимое рег. DIV_COUNTER
    decfsz DIV_COUNTER, F ;Уменьшить счетчик DIV_COUNTER на 1
```

```

goto div16_loop      ;Если счетчик не равен нулю, то приступить к
                    ;расчету следующего разряда результата деления
rlf ZAEHLER_L, F      ;Все значение числителя сдвинуть влево,
rlf ZAEHLER_M, F      ;чтобы в числителе содержался
rlf ZAEHLER_H, F      ;правильный результат
return               ;Возврат из подпрограммы

```

После возвращения в основную программу рассчитанное значение может быть взято из трех регистров для хранения числителя ZAEHLER_L, ZAEHLER_M и ZAEHLER_H. Если для последующих вычислений нужен остаток, то он может копироваться из соответствующих регистров остатка REST_L и REST_H.

11.4. Отображение расчетной мощности

После того как рассмотрены подпрограммы для выполнения умножения и деления, можно осуществить отображение расчетных значений. Напряжение и ток рассчитываются и выводятся на индикатор. Перед воспроизведением напряжение умножается на 100, чтобы смогли отображаться 2 разряда после запятой. Это же самое проделывается с оцифрованным значением тока. Если оба значения перемножат между собой, то получают значение мощности, которое будет соответственно в 10000 раз больше, чем настоящая мощность. Вследствие этого нужно выводить мощность с 4 разрядами после запятой. Для отображения на индикаторе нужно определять отдельные разряды. Этого можно достичь многократным делением на 10. После каждого деления результат сохраняется в регистре. Следующая часть основной программы показывает, как могут рассчитываться отдельные разряды. Поскольку программа относительно объемна, то здесь приведены только фрагменты из программы P-R-Messung. Полная работоспособная программа находится на прилагаемом к книге компакт-диске.

```

movlw 0x00           ;Задать вывод символа в поз. 1
                    ;во 2-й строке (=0x80 + 0x40)
call SchreibeDispKommando ;Передача команды для установки
                    ;адреса оперативной памяти индикатора
movlw A'P'           ;Загрузить символ "P" в рег. W для
                    ;вывода во 2-й строке в 1-й позиции
call SchreibeDispDaten ;Выполнить передачу символа
movlw A'='           ;Загрузить символ "=" в рег. W и
call SchreibeDispDaten ;выполнить передачу символа для вывода
                    ;его во 2-й строке во 2-й позиции
call Mull6           ;Вычисление мощности
                    ;Оба множителя находятся в
por                  ;регистрах MUL_1_H/MUL_1_L
                    ;и MUL_2_H/MUL_2_L. Поскольку значение тока

```

<code>movf MUL_E_3, W</code>	<code>;и значение напряжения ранее умножались</code>
<code>movwf ZAEHLER_H</code>	<code>;каждый раз на 100, то значение мощности</code>
<code>movf MUL_E_2, W</code>	<code>;в трех регистрах MUL_E_3 до MUL_E_1</code>
<code>movwf ZAEHLER_M</code>	<code>;будет храниться 10 000-кратное значение</code>
<code>movf MUL_E_1, W</code>	<code>;Чтобы поступало правильное значение,</code>
<code>movwf ZAEHLER_L</code>	<code>;значение мощности нужно многократно</code>
	<code>;разделить на 10. Для этого в регистры</code>
	<code>;числителя сначала загружаются</code>
	<code>;результаты умножения.</code>
<code>clrf NENNER_H</code>	<code>;Определение отдельных разрядов</code>
<code>movlw d'10'</code>	<code>;результата вычисления мощности</code>
<code>movwf NENNER_L</code>	<code>;делением на 10</code>
<code>call Div16</code>	<code>;Выполнение первого деления,</code>
<code>movf REST_L, W</code>	<code>;чтобы определить младший разряд.</code>
	<code>;Остаток деления – это значение</code>
<code>movwf STELLE_6</code>	<code>;для 4-го разряда после запятой, который</code>
	<code>;сохраняется в регистре STELLE_6</code>
<code>call Div16</code>	
<code>movf REST_L, W</code>	
<code>movwf STELLE_5</code>	<code>;Сохранение 3-го разряда после запятой</code>
<code>call Div16</code>	
<code>movf REST_L, W</code>	
<code>movwf STELLE_4</code>	<code>;Сохранение 2-го разряда после запятой</code>
<code>call Div16</code>	
<code>movf REST_L, W</code>	
<code>movwf STELLE_3</code>	<code>;Сохранение 1-го разряда после запятой</code>
<code>call Div16</code>	
<code>movf REST_L, W</code>	
<code>movwf STELLE_2</code>	<code>;Сохранение 1-го разряда</code>
	<code>;перед запятой (разряд единиц)</code>
<code>call Div16</code>	
<code>movf REST_L, W</code>	
<code>movwf STELLE_1</code>	<code>;Сохранение 2-го разряда</code>
	<code>;перед запятой (разряд десятков)</code>

В этом фрагменте программы на индикаторе сначала отображаются два символа для обозначения мощности (P=). Затем выполняется расчет значения мощности после вызова подпрограммы `Mul16` командой `call Mul16`. После выполнения измерения тока и напряжения их значения, умноженные каждый на 100, записываются в соответствующие регистры для выполнения процедуры умножения. После умножения уже 10000-кратное значение мощности будет храниться в четырех регистрах от `MUL_E_4` до `MUL_E_1`. Правда, в данном случае регистр `MUL_E_4` не используется, т. к. значение для мощности может

быть максимум 250000. Действительно, напряжение может изменяться от 0 до 5 В (после измерения и умножения на 100 от 0 до 500), а ток — от 0 и 5 А (соответственно от 0 до 500). Поскольку после умножения 500×500 в итоге получается значение равное 250000_{10} (111101000010010000_2). Поэтому для хранения такого результата вполне достаточно 18 битов, т. е. три 8-разрядных регистра. После этого содержимое этих трех регистров с результатом умножения копируется в регистры для хранения числителя. Затем для выполнения деления в регистр знаменателя загружается значение "10". Когда вся подготовка будет закончена, значение мощности может последовательно делиться на 10. После каждого деления остаток копируется в соответствующий регистр соответствующего разряда. После 6 делений десятичное значение результата будет находиться в регистрах `STELLE_1`—`STELLE_6`. Регистры `STELLE_1` и `STELLE_2` содержат оба разряда перед запятой, а регистры от `STELLE_3` до `STELLE_6` соответствуют четырем разрядам после запятой.

Теперь отдельные разряды для значения мощности известны и могут быть отображены на жидкокристаллическом индикаторе. При выводе информации на индикатор нужно учитывать, что отображать следует только три значащих разряда. Таким образом, лишние нули не должны выводиться, например, при отображении 00,0100 Вт. Чтобы гарантировать это, в разрядах перед запятой выполняется проверка, равны ли они 0. Если это происходит, то разряд на дисплее не показывается и начинается проверка следующего разряда. Поскольку мощность рассчитывалась с учетом 2 разрядов перед запятой и 4 разрядов после, то запятая на дисплее показывается после второй проверки в любом случае. Например, если мощность имеет значение 23 мВт, то на дисплее будет отображаться "0.23 W".

<code>prüfe_Stelle1_P</code>	;Для отображения на индикаторе
<code>movlw d'3'</code>	;должны использоваться только 3 разряда
	;При этом результат не округляется!
<code>movwf STELLE_CNT</code>	;Загрузить счетчик необходимым количеством
	;разрядов
<code>movlw 0xFF</code>	
<code>andwf STELLE_1, W</code>	;Проверить, является ли нулем крайний левый
<code>btfsc STATUS, Z</code>	;разряд, поскольку начальные (лишние) нули
	;отображаться не должны
<code>goto prüfe_Stelle2_P</code>	
<code>schreibe_Stelle1_P</code>	;Крайний левый разряд был больше чем 0
<code>movlw 0x30</code>	;Преобразовать десятичный разряд в код ASCII
<code>addwf STELLE_1, W</code>	
<code>call SchreibeDispDaten</code>	;Отобразить разряд на индикаторе
<code>decfsz STELLE_CNT</code>	;Поскольку был вывод разряда, следует
	;уменьшить значение счетчика разрядов

```

goto schreibe_Stelle2_P ;Если значение счетчика > 0,
                        ;выводится следующий разряд
goto schreibe_Einheit_P ;Все разряды выведены на индикатор
                        ;Надо начать отображение единицы измерения

pruefe_Stelle2_P
    movlw 0xFF
    andwf STELLE_2, W    ;Проверить, равен ли второй разряд 0
    btfsc STATUS, Z
    goto pruefe_Stelle3_P ;если разряд = 0, то проверяется третий
                        ;разряд
schreibe_Stelle2_P      ;2-й разряд > 0 и его надо отобразить
    movlw 0x30           ;Преобразовать десятичный разряд в код ASCII
    addwf STELLE_2, W
    call SchreibeDispDaten
    movlw A'.'           ;Мощность рассчитывалась с 4 разрядами
    call SchreibeDispDaten ;после запятой, поскольку ток и
    decfsz STELLE_CNT    ;напряжение ранее умножались на 100
    goto schreibe_Stelle3_P ;Поэтому десятичная точка
    goto schreibe_Einheit_P ;должна выводиться после этого разряда
                        ;Затем выводится 3-й разряд
                        ;или единица измерения,
                        ;если счетчик разрядов равен 0
pruefe_Stelle3_P      ;Если разряд 2 был 0 и
    movlw A'.'           ;отсутствует разряд перед запятой, то
                        ;десятичная точка будет первым знаком
    call SchreibeDispDaten ;Вывод на индикатор десятичной точки
schreibe_Stelle3_P     ;Вывод всех цифр с 3-го разряда
    movlw 0x30           ;Преобразовать десятичный разряд в код ASCII
    addwf STELLE_3, W
    call SchreibeDispDaten
    decfsz STELLE_CNT    ;Проверить, выведены ли уже 3 цифры
    goto schreibe_Stelle4_P ;Если да, то вывод символа единицы измерения,
    goto schreibe_Einheit_P ;иначе выводят 4-й разряд
schreibe_Stelle4_P     ;Вывод 2-го разряда после запятой
    movlw 0x30           ;Преобразовать десятичный разряд в код ASCII
    addwf STELLE_4, W
    call SchreibeDispDaten
    decfsz STELLE_CNT    ;Если есть еще 3-й разряд после запятой,
    goto schreibe_Stelle5_P ;то перейти к выводу 5-го разряда
    goto schreibe_Einheit_P ;Перейти к отображению единицы измерения,
                        ;если счетчик разрядов равен 0
schreibe_Stelle5_P     ;Вывод 3-го разряда после запятой
    movlw 0x30           ;Преобразовать десятичный разряд в код ASCII

```



```

addwf STELLE_5, W      ;Только максимум 3 значащих разряда
                        ;должны выводиться, поэтому 5-й
call SchreibeDispDaten  ;разряд — это последний отображаемый разряд
                        ;6-й разряд больше не должен
                        ;учитываться.
schreibe_Einheit_P      ;После последнего значащего разряда
movlw A'W'              ;выводится единица измерения мощности,
call SchreibeDispDaten ;а именно символ "W"

```

Для упрощения программы выводится только последний разряд без округления. Поэтому 6-й разряд не учитывается при отображении и просто отбрасывается. После выдачи 3 значащих разрядов выводится символ "W" (ватт), как единица измерения электрической мощности.

11.5. Отображение рассчитанного сопротивления

После вывода на индикатор мощности можно приступить к отображению значения сопротивления. Как и при измерении мощности, при определении сопротивления также представляют интерес разряды после запятой. Чтобы определить сопротивление с 3 разрядами после запятой, значение напряжения умножается на коэффициент 1000. Чтобы сохранить 1000-кратное значение напряжения, требуется как минимум 19-разрядный регистр ($500 \times 1000 = 500000_{10} = 1111010000100100000_2$). Однако представленная ранее подпрограмма `Mul16` поддерживает только умножение двух 16-битных значений. Для решения этой задачи имеются два варианта. С одной стороны, можно расширить ранее написанную подпрограмму таким образом, чтобы сделать возможным умножение регистров с большей разрядностью. Другая возможность — написать новую подпрограмму, которую можно будет использовать и для других приложений. Поскольку в программах часто требуется умножение значения на 10, то лучше написать собственную подпрограмму для умножения на 10. Если нужно умножить число на известный коэффициент (здесь 10), то можно существенно оптимизировать подпрограмму и вследствие этого значительно сэкономить на вычислениях. Операцию умножения на 10 можно разделить, например, на простые команды сдвига и сложения ($10 = 2 \times 2 \times 2 + 1 + 1$). Поэтому в следующей подпрограмме выполняются три сдвига влево содержимого регистров, что означает умножение на 8. Затем значение должно дважды суммироваться, чтобы в итоге реализовать умножение на 10. Следующая подпрограмма `Mul_x10` требует три 8-разрядного регистра для множимого (24 бита) (`MUL_X10_H`, `MUL_X10_M`, `MUL_X10_L`) и три аналогичных регистра для результата (`MUL_X10_H_ERG`, `MUL_X10_M_ERG`, `MUL_X10_L_ERG`).

```

mul_x10
    movf MUL_X10_L, W           ;Копировать входные данные в регистры
    movwf MUL_X10_L_ERG        ;MUL_X10_H_ERG, MUL_X10_M_ERG, MUL_X10_L_ERG
    movf MUL_X10_M, W          ;для временного хранения промежуточных
    movwf MUL_X10_M_ERG        ;результатов
    movf MUL_X10_H, W
    movwf MUL_X10_H_ERG
    movlw d'3'                 ;Содержимое регистров временного хранения
    movwf COUNTER              ;результата для выполнения операции умножения
                                ;на 8 должно 3 раза сдвигаться влево

mul_x2_loop
    bcf STATUS, C              ;Очистить бит переноса C, чтобы он
                                ;не повлиял операции умножения

    rlf MUL_X10_L_ERG, F       ;Умножение на 2
    rlf MUL_X10_M_ERG, F
    rlf MUL_X10_H_ERG, F
    decfsz COUNTER, F          ;Уменьшить на 1 счетчик количества умножений
    goto mul_x2_loop          ;Если значение счетчика не 0, то повторить
                                ;операцию умножения на 2

    movlw d'2'                 ;Теперь значение умножено на 8
    movwf COUNTER              ;Задать в счетчике количество операций
                                ;сложения числовых значений равное 2

add_loop
    movf MUL_X10_L, W           ;Добавить младший байт числового значения
    addwf MUL_X10_L_ERG, F
    btfsc STATUS, C            ;Проверить, был ли перенос и, если был, добавить
    incf MUL_X10_M_ERG, F      ;1 к содержимому следующего рег-ра результата
    movf MUL_X10_M, W          ;Добавить средний байт числового значения
    addwf MUL_X10_M_ERG, F
    btfsc STATUS, C            ;Проверить, был ли перенос и, если был, добавить
    incf MUL_X10_H_ERG, F      ;1 к содержимому следующего рег-ра результата
    movf MUL_X10_H, W          ;Добавить старший байт числового значения
    addwf MUL_X10_H_ERG, F
    decfsz COUNTER, F          ;Уменьшить значение счетчика и проверить его
    goto add_loop             ;Повторить операцию сложения, если значение в
                                ;счетчике не равно 0

    movf MUL_X10_L_ERG, W      ;Если равно 0, т. е. 2 операции уже выполнены,
    movwf MUL_X10_L           ;то скопировать результат умножения на 10 из
    movf MUL_X10_M_ERG, W      ;регистров, предназначенных для временного
    movwf MUL_X10_M           ;хранения, в регистры MUL_X10_H, MUL_X10_M и
    movf MUL_X10_H_ERG, W      ;MUL_X10_L, в которых изначально было множимое
    movwf MUL_X10_H
    return

```

После выполнения подпрограммы результат умножения на 10 снова копируется в регистр, в котором изначально хранилось множимое. Таким образом, троекратным вызовом подпрограммы можно реализовать умножение на 1000.

Перед началом выполнения вычислений нужно заранее скопировать значения измеренного напряжения и тока в соответствующие регистры. Для выполнения последующего деления значение, соответствующее измеренному току, копируется в регистры для хранения знаменателя, а значение напряжения сначала копируется в регистры для умножения на 10.

```

movf STROM_L, W           ;Значение тока поступает в 16-разрядный регистр
movwf NENNER_L            ;и копируется в оба 8-разрядных
movf STROM_H, W           ;регистры для хранения знаменателя
movwf NENNER_H
movf SPG_L, W             ;Чтобы иметь возможность отображать небольшое
movwf MUL_X10_L           ;сопротивление (менее 1 Ом), значение
movf SPG_H, W             ;напряжения должно умножаться на коэфф. 1000
movwf MUL_X10_M           ;Таким способом можно рассчитать
clrf MUL_X10_H            ;3 десятичных разряда после запятой
                           ;Для этого значение напряжения
                           ;копируется в регистры для умножения
                           ;MUL_X10_L, MUL_X10_M, MUL_X10_H

nop
call Mul_x10              ;Выполнить 3-кратное умножение напряжения
call Mul_x10              ;(10 * 10 * 10 = 1000)
call Mul_x10              ;Самое маленькое сопротивление:
                           ;0,001 В / 5,00 А = 0,002 Ом

nop                        ;Самое большое сопротивление:
                           ;5.00 В / 0.01 А = 500 Ом

movf MUL_X10_L_ERG, W     ;Копировать 1000-кратное значение напряжения
movwf ZAEHLER_L           ;для выполнения деления
movf MUL_X10_M_ERG, W     ;Поскольку значение напряжения может быть
movwf ZAEHLER_M           ;максимум 500 000, то поэтому оно поступает
movf MUL_X10_H_ERG, W     ;в 24-разрядный регистр для выполнения
movwf ZAEHLER_H           ;операции деления

```

После вычисления 1000-кратного напряжения содержимое трех регистров MUL_X10_x_ERG копируется в соответствующие три регистра, предназначенные для хранения числителя. В программе снова и снова встречаются команды nop. Они не обязательны, но дают возможность устанавливать точку останова после вычислений.

Теперь подготовка для вычисления сопротивления закончена, и значение сопротивления может быть рассчитано путем вызова подпрограммы Div16.

В ходе вычисления отдельные десятичные разряды, как и при вычислении мощности, определяются посредством неоднократного деления на 10.

```
call Div16      ;Вызов подпрограммы деления регистров числителя
  пор          ; (напряжение) на регистры знаменателя
               ; (ток) для получения значения сопротивления
  clrf NENNER_H ;Результат деления находится в регистрах числителя
  movlw d'10'   ;Чтобы определить отдельные десятичные
  movwf NENNER_L ;разряды значения сопротивления, результат
  call Div16     ;последовательно делится на 10. Соответствующий
  movf REST_L, W ;остаток пишется в регистр для
               ;соответствующего разряда.
  movwf STELLE_6 ;3-й десятичный разряд после запятой
  call Div16
  movf REST_L, W
  movwf STELLE_5 ;2-й десятичный разряд после запятой
  call Div16
  movf REST_L, W
  movwf STELLE_4 ;1-й десятичный разряд после запятой
  call Div16
  movf REST_L, W
  movwf STELLE_3 ;1-й десятичный разряд перед запятой (единицы)
  call Div16
  movf REST_L, W
  movwf STELLE_2 ;2-й десятичный разряд перед запятой (десятки)
  call Div16
  movf REST_L, W
  movwf STELLE_1 ;3-й десятичный разряд перед запятой (сотни)
```

После выполнения этой части программы разряды перед запятой будут храниться в регистрах STELLE_1—STELLE_3, а 3 разряда после запятой — в регистрах STELLE_4—STELLE_6. Также как и при отображении на индикаторе мощности, в этом случае, при отображении сопротивления, должны выводиться только три значащих разряда. Перед выводом цифрового значения сопротивления должны отображаться символы "R = ". Они выводятся во второй строке, начиная с десятой позиции. Вслед за этим проверяется, имеются ли в десятичном значении сопротивления стоящие впереди нули, которые не должны отображаться на индикаторе. Для этого начинают проверку со старшего разряда (STELLE_1). Если проверенный разряд не равен нулю, то он выводится на индикатор и счетчик для подсчета значащих разрядов уменьшается на единицу. Если все три значащих разряда выведены, то в счетчике будет храниться 0, после чего отображается символ для единицы измерения сопротивления " Ω " (Ом).


```

decfsz STELLE_CNT          ;Если счетчик еще больше чем 0,
goto schreibe_Stelle3      ;перейти к следующему разряду
goto schreibe_Einheit      ;В противном случае перейти к выводу
                             ;единицы измерения

pruefe_Stelle3
    movlw 0xFF              ;Если все предыдущие разряды были 0,
    andwf STELLE_3, W       ;здесь проверяется, является ли нулем
                             ;разряд единиц перед запятой
    btfsc STATUS, Z         ;Если он также нуль, то сопротивление
    goto pruefe_Stelle4     ;меньше чем 1 Ом и выполняется переход
                             ;к рассмотрению следующего разряда

schreibe_Stelle3
    movlw 0x30              ;Преобразовать десят. разряд единиц в ASCII
    addwf STELLE_3, W
    call SchreibeDispDaten  ;Выполнить передачу разряда единиц на
                             ;индикатор
    movlw A'.'              ;После разряда единиц следуют только лишь
    call SchreibeDispDaten  ;разряды после запятой,
    decfsz STELLE_CNT       ;поэтому вначале выводится десятичная точка
                             ;Если 3 значащих разряда уже выведены, то
    goto schreibe_Stelle4   ;перейти к выводу единицы измерения
    goto schreibe_Einheit   ;Если нет, перейти к
                             ;обработке 4-го разряда

pruefe_Stelle4
    movlw A'.'              ;Если все десятичные разряды перед запятой
    call SchreibeDispDaten  ;были равны 0, то имеются только разряды
                             ;после запятой и поэтому прежде всего
                             ;выводится десятичная точка
    schreibe_Stelle4        ;С 4-го разряда все цифры
    movlw 0x30              ;должны выводиться, поэтому
    addwf STELLE_4, W       ;делается проверка,
                             ;является ли разряд 0
    call SchreibeDispDaten  ;Отобразить первый разряд после запятой
    decfsz STELLE_CNT       ;Проверить, должны ли выводиться следующие
    goto schreibe_Stelle5   ;разряды или же нужно перейти к выводу
    goto schreibe_Einheit   ;единицы измерения

schreibe_Stelle5
    movlw 0x30              ;Преобразовать 2-й разряд после запятой
                             ;в код ASCII
    addwf STELLE_5, W
    call SchreibeDispDaten  ;Отобразить второй разряд после запятой
    decfsz STELLE_CNT       ;Уменьшить счетчик для значащих разрядов
    goto schreibe_Stelle6   ;Таким образом проверяется,
    goto schreibe_Einheit   ;должна ли выводиться еще одна цифра

```

```
schreibe_Stelle6
    movlw 0x30                ;Преобразовать 3-й разряд после запятой
                                ;в код ASCII

    addwf STELLE_6, W
    call SchreibeDispDaten    ;Отобразить последний разряд после запятой
                                ;это значение для мОм (миллиом)

schreibe_Einheit
    movlw 0xCF                ;Вывод символа единицы измерения
                                ;сопротивлений
    call SchreibeDispKommando ;Таким образом, последним выводится символ
                                ;"Ω" (Ом)

    movlw 0x1E
    call SchreibeDispDaten
```

Десятичная точка должна выводиться после разряда единиц. Если все разряды перед запятой были нулями, то десятичная точка является первым символом, который представляется. После десятичной точки все цифры могут выводиться без следующей проверки.

Теперь по окончании выдачи значения сопротивления основная программа может быть запущена с самого начала.

12. Передача данных посредством последовательного интерфейса

В предыдущих примерах микроконтроллер всегда использовался как самостоятельный узел, сообщая сведения пользователю только с помощью жидкокристаллического индикатора или светодиодов. Однако часто требуется для обмена данными подключать микроконтроллер к *персональному компьютеру* (ПК). Может возникнуть, например, случай, когда микроконтроллер только накапливает результаты измерения, которые должны передаваться в персональный компьютер для дальнейшей обработки с более высокой вычислительной мощностью. С помощью значительно более высокой вычислительной производительности компьютера можно представлять на мониторе обработанные результаты измерения в виде графиков. Другой пример, при котором наоборот персональный компьютер должен передавать данные в микроконтроллер, — это пример управления внешними аппаратными средствами. В таком случае на персональном компьютере программируется графический интерфейс, с помощью которого управляют внешним устройством, а подключают аппаратные средства к персональному компьютеру посредством *последовательного интерфейса*. Теперь есть возможность, например, управлять по радио регулированием температуры с помощью персонального компьютера. Микроконтроллеру только лишь сообщаются параметры желаемой температуры, а он далее сам обеспечивает регулирование.

Имеются различные последовательные интерфейсы, которые отличаются тем, как они передают данные. Самые известные интерфейсы, имеющиеся в наличии на персональном компьютере по умолчанию, — это интерфейсы *USB* (Universal Serial Bus) и *RS-232* (от англ. Recommended Standard). Интерфейс *USB* является более сложным, но с его использованием можно достичь высоких скоростей передачи. Интерфейс *RS-232* применяется в большинстве случаев лишь в настольных компьютерах и в очень немногих портативных компьютерах. К сожалению, интерфейс *USB* является относительно комплекс-

ным в управлении и поэтому для многих маленьких микроконтроллеров, в том числе и PIC16F876A, его применение невыгодно. Протокол и управление посредством интерфейса RS-232 осуществляются очень просто, и этот интерфейс может также легко быть реализован на основе микроконтроллера. В большинстве случаев невысокой скорости передачи посредством интерфейса RS-232 совершенно достаточно, т. к. должен передаваться только небольшой объем данных. Обычно скорость интерфейса RS-232 составляет примерно 10—100 кбод (англ. baud). Для увеличения скорости передачи требуется, однако, установка в некоторых персональных компьютерах дополнительной съемной платы. Если нужно осуществлять связь между микроконтроллером и портативным компьютером, который не располагает интерфейсом RS-232, то требуется присоединять порт USB через адаптер. Электроника в этом адаптере предоставляет в распоряжение компьютеру так называемый виртуальный *COM-порт* (англ. Communication Port) и управляет обменом по последовательному интерфейсу RS-232. Для пользователя все выглядит тогда таким образом, как будто персональный компьютер располагает интерфейсом RS-232.

12.1. Последовательный интерфейс RS-232

Прежде чем начать программирование последовательного интерфейса, нужно знать еще кое-что о виде передачи данных, чтобы соответствующим образом можно было бы настраивать аппаратные средства. В последовательном интерфейсе RS-232 данные передаются последовательно, т. е. по линии связи каждый бит посылается поочередно после передачи предыдущего бита. В дальнейшем будет рассмотрен только самый простой вид обмена, а не все возможности стандарта RS-232. Для самой простой передачи между персональным компьютером и микроконтроллером требуются только лишь 3 линии: одна для передачи TX (Transmit), одна для приема RX (Receive) и одна для общей шины GND (Ground). Если же данные передаются только в одном направлении, то будет достаточно только двух линий.

12.1.1. Подключение через последовательный интерфейс

К сожалению, в интерфейсе RS-232 передаваемые данные должны иметь более высокие уровни напряжения, чем уровни ТТЛ (транзисторно-транзисторной логики). Для передачи отдельных битов используются положительные и отрицательные напряжения. Причем уровень между +3 В и +15 В интерпретируется как низкий логический уровень (Low), а напряжения между

–3 В и –15 В как высокий (High). Как правило, для передачи используются напряжения +12 В и –12 В. Сигналы такого уровня не могут непосредственно генерироваться микроконтроллером, поскольку его сигналы имеют напряжения только между 0 и 5 В. Поэтому необходим еще дополнительный преобразователь для приведения напряжения к требуемому уровню. Типичная микросхема такого преобразователя, например, MAX232A производится компанией Maxim. Эта микросхема с помощью внутреннего преобразователя напряжения преобразует уровни сигналов последовательного интерфейса в ТТЛ-уровни и наоборот. Подобные микросхемы изготавливаются многими производителями и имеют различное количество каналов и разное напряжение питания.

Как правило, в персональном компьютере имеется 9-контактный разъем Sub-D (вилка), к которому может присоединяться кабель для последовательной передачи данных. На стороне подключаемых аппаратных средств кабель имеет соответствующий обратный 9-контактный разъем Sub-D (гнездо). Схема подключения микроконтроллера к персональному компьютеру через последовательный интерфейс показана на рис. 12.1.

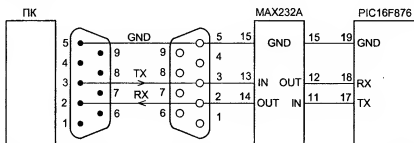


Рис. 12.1. Подключение микроконтроллера посредством последовательного интерфейса

Во многих случаях выбирают именно этот вариант подключения, т. к. при этом можно быстро собрать необходимую электрическую схему подключения аппаратных средств и достаточно просто запрограммировать соответствующее программное обеспечение.

12.1.2. Протокол интерфейса RS-232

Чтобы данные могли последовательно передаваться по линии, для них должен быть определен момент начала и конца слова. Имеются различные виды протоколов передачи по интерфейсу. Тем не менее в любом случае передача начинается со стартового бита и заканчивается стоповым битом. Формат данных и скорость передачи, используемые в передатчике, должны быть установлены точно такими же, как и в приемнике, поскольку иначе не может

быть обеспечена надежная передача. Поэтому при проблемах передачи сначала нужно проверять все настройки.

При самом простом способе передачи применяется 1 стартовый бит, 8 битов данных и 1 стоповый бит. Если же требуется наиболее достоверная передача, то может быть добавлен еще бит четности, который передается после слова данных. Бит четности предназначен для защиты от ошибок, который определяет отдельные ошибки при передаче данных. Если при передаче будет искажен один бит, и соответственно неверно определен, то эта ошибка может быть определена. Если же будут искажены два или большее количество битов, то с помощью бита четности определить такие ошибки уже не представляется возможным. В случае использования бита четности можно выбирать между проверкой на четность и нечетность. При проверке на четность (Even-Parity) в бите четности будет логический 0, если сумма единиц в слове данных четная, или будет логическая 1, если сумма нечетная. При проверке на нечетность — все в точности наоборот. В табл. 12.1 приведены все возможные состояния.

Таблица 12.1. Значение бита контроля четности

Сумма единиц в слове данных	Проверка на четность (Even-Parity)	Проверка на нечетность (Odd-Parity)
Четная	0	1
Нечетная	1	0

Во многих случаях от передачи бита четности отказываются, т. к. в этом случае требуются дополнительные затраты на проверку или установку бита соответствующим образом.

После того как определен формат для передачи данных, нужно устанавливать еще скорость, с которой должны передаваться данные. Поскольку общая линия с тактовым сигналом отсутствует, то перед началом обмена данными нужно сообщить передатчику и приемнику, с какой скоростью будет осуществляться передача. Поэтому такую передачу называют *асинхронной*. Скорость тактирования или скорость передачи указывается в бодах. Нельзя путать эту скорость со скоростью передачи данных, т. к. при этом речь идет о количестве полезных данных. При скорости передачи 19200 бод или 19,2 кбод уровень сигнала изменяется 19200 раз в секунду. Так как для передачи 1 байта еще соответственно требуется стартовый и стоповый бит и скорость передачи данных составляет максимум 8/10 скорости передачи. Из этого следует, что в этом примере могут передаваться максимум 15360 бит в секунду или 1920 байт в секунду. Поскольку в этом случае имеется в виду

асинхронная передача данных, то стартовый бит может посылаться в любое время. Приемник должен тогда во время обмена считывать данные с той же самой скоростью, с которой они посылаются передатчиком. Процедуру передачи данных без бита четности можно представить на рис. 12.2. В данном случае передается латинская буква "M" в ASCII-коде ("M" = 0x4D).

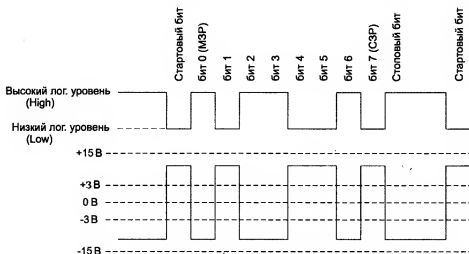


Рис. 12.2. Передача латинской буквы М

Как можно видеть на рис. 12.2, в режиме простоя на линии всегда имеется отрицательный уровень сигнала, если никакие данные не передаются. Передача начинается со стартового бита, который переключает отрицательный уровень (высокий логический уровень) на положительный уровень (низкий логический уровень). Затем посылаются 8 битов данных. Если для передачи требуется использовать бит четности, то он вставляется за СЗР (MSB) (бит 7). В конце данных посылается стоповый бит, который снова возвращает уровень сигнала в состояние простоя. После этого может выполняться передача следующего слова, которая снова начинается со стартового бита.

12.2. Программное обеспечение для передачи данных

Чтобы передавать данные в микроконтроллер или принимать их от него, требуется специальное программное обеспечение на персональном компьютере, которое может отображать эти данные. В простейшем случае, как правило, в операционной системе имеется программа для работы на терминале. Если используется ПО Windows, то это программа HyperTerminal, которую можно

найти из меню **Пуск** → **Программа** → **Стандартные** → **Связь** → **HyperTerminal**. Кроме того, в Интернете имеется огромное количество аналогичных бесплатных программ.

После запуска программы нужно указывать несколько установок для нового подключения. Все сделанные установки можно изменять также снова в более поздней момент в соответствующем меню. В процессе настроек делается запрос о том, какой COM-порт нужно использовать в формате передачи. Для апробирования представленного далее примера в программе HyperTerminal должны быть сделаны следующие установки:

COM-порт:	Зависит от применяемого персонального компьютера, в большинстве случаев это COM1
Скорость (бит/с):	19 200
Биты данных:	8
Четность:	Нет
Стоповые биты:	1
Управление потоком:	Нет

Далее при готовом подключении должна сразу же выполняться передача и прием данных. В окне программы терминала отображаются только принятые символы. Если нажать на клавишу, то символ на дисплее не будет отображен, как это обычно происходит в программе редактора. Однако при имеющемся подключении символ будет передан с помощью последовательного интерфейса. Все же, чтобы увидеть этот символ, можно сделать маленький трюк. Для этого в разьеме Sub-D (гнезде) нужно объединить вывод 2 (RX) с выводом 3 (TX). Таким образом, посланный символ от программы работы на терминале сразу снова принимается и представляется на дисплее. Это является также простой возможностью тестирования последовательного интерфейса. Если сделать это короткое замыкание в месте, в котором обычно подключен микроконтроллер, то можно быть уверенным, что данные безошибочно передавались до этого места. Если после этой проверки передача функционирует не так, как ожидалось, то это с высокой долей вероятности зависит от программного обеспечения микроконтроллера.

12.3. Применение интерфейса USART

Микроконтроллер PIC16F876A располагает модулем, который существенно упрощает передачу данных с помощью последовательного интерфейса. Здесь речь идет о модуле универсального синхронно-асинхронного приемопередатчика УСАПП или иначе *USART* (Universal Synchronous Asynchronous

Receiver Transmitter). Этот модуль, судя по названию, может реализовать как синхронную, так и асинхронную передачу данных в обоих направлениях. Чтобы выполнять обмен в распоряжении этого модуля, имеются 5 специальных регистров. Здесь имеется в виду прежде всего регистр состояния и управления передатчиком модуля USART — TXSTA и регистр состояния и управления приемника — RCSTA, в которых устанавливается вид передачи и с помощью которых может запрашиваться состояние линии связи. Установка скорости обмена в бодах осуществляется с помощью регистра SPBRG, а в регистрах TXREG и RCREG сохраняются передаваемые и соответственно принимаемые данные.

12.3.1. Установка скорости в бодах

Для установки скорости, с которой должны передаваться данные, в микроконтроллере PIC16F876A имеется генератор скорости обмена информацией в бодах. Этот задающий генератор заботится о том, чтобы данные передавались с той же самой скоростью, которая задана в персональном компьютере. Чтобы установить скорость обмена в бодах, нужно записать соответствующее значение в регистр SPBRG. Это значение можно взять из таблицы в техническом описании микроконтроллера или рассчитать лучшее значение по приведенным там же формулам. Поскольку тактовая частота микроконтроллера, как правило, имеет не целочисленное кратное желаемой скорости обмена в бодах значение, из-за этого может возникать определенная погрешность. При настройке параметров это нужно обязательно учитывать. При низких значениях скорости обмена эта ошибка менее критична, чем при более высоких скоростях. Поэтому всегда нужно пытаться выбирать оптимальную настройку, при которой погрешность была бы минимальной. В представленных далее примерах PIC-контроллер синхронизируется тактовой частотой 4 МГц и, используя асинхронный режим работы, микроконтроллер должен передавать данные со скоростью 19,2 кбод. При настройке скорости еще нужно обращать внимание на правильность установки бита BRGH в регистре TXSTA. Для относительно низкой скорости обмена этот бит должен быть сброшен ($BRGH = 0$), а для высокой скорости — установлен в 1. При некоторых низких скоростях передачи может случиться так, что когда бит $BRGH = 1$, то также будет незначительная погрешность. Поэтому от случая к случаю нужно тщательно проверять установки. В следующих примерах бит BRGH был установлен в значение логической 1. В этом случае согласно таблице, приведенной в технической документации микроконтроллера, в регистр SPBRG должно быть записано значение 12₁₀. Если требуется устанавливать иную, от стандарта скорость обмена в бодах или использовать тактовую частоту, которая не указана в таблице, то можно воспользоваться следующими формулами для вычисления, в которых скорость обмена в бодах обозначена как $V_{\text{обмена}}$:

Для BRGH = 1 (Высокая скорость):

$$V_{\text{обмена}} = \frac{F_{\text{osc}}}{16 \cdot (\text{SPBGR} + 1)} \quad \text{SPBGR} = \frac{F_{\text{osc}}}{V_{\text{обмена}} \cdot 16} - 1 \quad (12.1)$$

Для BRGH = 0 (Низкая скорость):

$$V_{\text{обмена}} = \frac{F_{\text{osc}}}{64 \cdot (\text{SPBGR} + 1)} \quad \text{SPBGR} = \frac{F_{\text{osc}}}{V_{\text{обмена}} \cdot 64} - 1 \quad (12.2)$$

12.3.2. Установка регистров TXSTA и RCSTA

Для передачи с помощью последовательного интерфейса применяется асинхронная передача данных и поэтому на биты для работы в синхронном режиме можно не обращать внимание. Расположение отдельных битов в управляющих регистрах TXSTA и RCSTA показано на рис. 12.3.

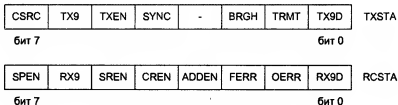


Рис. 12.3. Регистры TXSTA и RCSTA

Чтобы, прежде всего, можно было бы использовать последовательный интерфейс, нужно разрешать последовательную передачу посредством бита SPEN в регистре RCSTA. Если этот бит устанавливается в значение логической 1, то выводы порта RC6 и RC7 будут подключены к модулю USART.

В асинхронном режиме биты CSRC и SREN не имеют значения и поэтому могут иметь любое значение. Чтобы активировать асинхронный режим обмена, бит SYNC должен быть сброшен в значение логического 0. С помощью битов TX9 и RX9 разрешается 9-битная (TX9 = 1) или 8-битная передача (TX9 = 0) и соответственно 9-битный (RX9 = 1) или 8-битный прием (RX9 = 0). При этом девятый бит может быть, например, битом четности. Однако он не определяется автоматически аппаратными средствами в микроконтроллере и поэтому должен рассчитываться программистом. Поскольку в данном случае речь идет о 8-битном микроконтроллере и девятый бит в регистрах отсутствует, то для этого применяют биты TX9D и RX9D. Чтобы передавать данные с помощью последовательного интерфейса, нужно установить бит TXEN в состояние логической 1 и таким образом разрешить передачу. Однако если даже этот бит будет установлен, то данные сразу же переданы не

будут. Передача данных не начнется до тех пор, пока в регистр данных TXREG не будут загружены новые данные. Чтобы проверить, полон ли регистр данных или пуст, нужно запросить бит TRMT, являющийся флагом очистки сдвигового регистра передатчика. Если бит установлен в 1, то регистр пуст и может записываться новыми данными. Если же этот бит сброшен (TRMT = 0), т. е. сдвиговый регистр полон, данные уже выводятся с помощью последовательного интерфейса и нужно ждать поступления новых данных.

Для приема данных нужно установить бит CREN в состояние логической 1. Во время передачи данные читаются последовательно во внутренний регистр и после успешного получения копируются в регистр RCREG. Если будет получен стоповый бит, то вызывается прерывание и данные можно забирать из регистра RCREG. Если же данные еще не считаны и уже будут переданы новые данные, то может произойти переполнение внутреннего буфера. Если это случается, то устанавливается бит OERR (Overrun Error — ошибка переполнения). В таком случае, прежде чем принимать новые данные, нужно прежде всего очистить этот бит. Для этого сначала сбрасывают, а затем снова устанавливают бит CREN, который соответственно запрещает или разрешает прием данных. После этого можно будет снова принимать данные и считывать их из регистра RCREG. Если же при приеме данных стоповый бит будет неверно распознан, то устанавливается бит FERR (Frame Error — ошибка кадра). Бит ADDEN, разрешающий детектирование адреса, позволяет принимать и распознавать 9-битные адреса. Если же этого не требуется, то девятый бит может использоваться в качестве бита четности. Для передачи посредством интерфейса RS-232 этот бит сбрасывают в 0.

12.4. Пример программы "Управление с помощью компьютера"

Передачу данных по последовательному интерфейсу RS-232 можно опробовать с представленной монтажной платой и с помощью приведенного примера программы. На приложенном к книге компакт-диске можно найти и просмотреть полный исходный программный код и изменить его по желанию. Пример показывает, как можно относительно просто управлять переключением светодиодов на монтажной плате с помощью терминальной программы, например, программы HyperTerminal. В зависимости от нажатия той или иной клавиши на клавиатуре персонального компьютера передается различный код, по которому можно определить, какая именно нажата клавиша.

Однако прежде чем передавать данные с помощью последовательного интерфейса, он соответствующим образом должен быть проинициализирован.

```

init_serial
    _BANK_1
    movlw D'12'           ;Задать скорость обмена в бодах, равную 19,2 кбод
    movwf SPBRG           ; (точное значение 19 231 бод)
    movlw B'00000100'     ;Установить 8-битный, высокоскоростной режим
    movwf TXSTA
    _BANK_0
    movlw B'10010000'     ;Разрешить прием последовательных данных
    movwf RCSTA

```

После инициализации начинается основной цикл, в котором циклически запрашивается, получены ли последовательные данные или было ли какое-нибудь нажатие на кнопку на монтажной плате.

```

main                               ;Начало основного цикла
    btfsc PIR1, RCIF              ;Проверить, имеются ли последовательные данные
    goto lese_RCREG               ;Приступить к чтению регистра приема данных RCREG
    btfss Taster_1                 ;Проверить, нажимали ли на кнопку 1
    goto taster1
    btfss Taster_2                 ;Проверить, нажимали ли на кнопку 2
    goto taster2
    btfss Taster_3                 ;Проверить, нажимали ли на кнопку 3
    goto taster3
    btfss Taster_4                 ;Проверить, нажимали ли на кнопку 4
    goto taster4
    goto main                     ;Заключить проверку принятых данных и
                                ;опрос нажатий на кнопки

```

При наличии последовательных данных и установленном бите прерывания основная программа переходит на метку `lese_RCREG`, по которой считываются данные и осуществляется запуск соответствующей процедуры. Если на клавиатуре персонального компьютера нажали на клавишу <1>, то должен включаться светодиод LED 1. При нажатии клавиш <2>, <3> или <4> включаются соответствующие светодиоды LED 2, LED 3 или LED 4.

```

lese_RCREG
    movf RCREG, W                ;Считать данные из регистра приемника в
                                ;регистр W
    movwf DATEN                  ;Временно сохранить данные в регистре DATEN
    movf DATEN, W                ;Получить в рег. W данные, сохраненные в рег. DATEN
    xorlw A'1'                   ;Сравнить данные с ASCII-кодом "1"
    btfsc STATUS, Z              ;Если была передана 1, то
    goto led1                    ;перейти на метку led1
    movf DATEN, W                ;Получить в рег. W данные, сохраненные в рег. DATEN

```

```

xorlw A'2'      ;Сравнить данные с ASCII-кодом "2"
btfsc STATUS, Z ;Если была передана 2, то
goto led2      ;перейти на метку led2
movf DATEN, W   ;Получить в рег. W данные, сохраненные в рег. DATEN
xorlw A'3'      ;Сравнить данные с ASCII-кодом "3"
btfsc STATUS, Z ;Если была передана 3, то
goto led3      ;перейти на метку led3
movf DATEN, W   ;Получить в рег. W данные, сохраненные в рег. DATEN
xorlw A'4'      ;Сравнить данные с ASCII-кодом "4"
btfsc STATUS, Z ;Если была передана 4, то
goto led4      ;перейти на метку led4
goto main

```

Если были нажаты другие цифры или буквы, то происходит возврат на начало основного цикла программы без выполнения каких-либо действий.

```

led1      ;Включить светодиод LED1
btfsc LED_1 ;Проверить, включен ли светодиод или нет
goto led1_aus
bsf LED_1   ;если светодиод выключен, включить его
goto main

led1_aus
bcf LED_1   ;если светодиод включен, то выключить
goto main

led2      ;Включить светодиод LED2
btfsc LED_2 ;Проверить, включен ли светодиод или нет
goto led2_aus
bsf LED_2   ;если светодиод выключен, включить его
goto main

led2_aus
bcf LED_2   ;если светодиод включен, то выключить
goto main

led3      ;Включить светодиод LED3
btfsc LED_3 ;проверить, включен ли светодиод или нет
goto led3_aus
bsf LED_3   ;если светодиод выключен, включить его
goto main

led3_aus
bcf LED_3   ;если светодиод включен, то выключить
goto main

led4      ;Включить светодиод LED4
btfsc LED_4 ;проверить, включен ли светодиод или нет
goto led4_aus
bsf LED_4   ;если светодиод выключен, включить его
goto main

```

```

led4_aus
    bcf LED_4      ;если светодиод включен, то выключить
    goto main

```

После включения или выключения соответствующего светодиода опять будет переход в основной цикл программы, а затем проверка состояния приемника и кнопок. Если на монтажной плате будет нажата кнопка, то основная программа в зависимости от нажатой кнопки перейдет на соответствующую метку `taster1`, `taster2`, `taster3` или `taster4`. При нажатии той или иной кнопки на персональный компьютер должен передаваться определенный текст: для кнопки 1 — "S1", для кнопки 2 — "S2", для кнопки 3 — "S3" и для кнопки 4 — "S4". Поскольку достаточно часто требуется выполнять передачу символов, то имеет смысл написать специальную подпрограмму для передачи символа с помощью последовательного интерфейса.

Подпрограмма `SendeZeichen` разрешает передачу и копирует посылаемый символ из регистра `W` в регистр данных передатчика `TXREG`. После копирования данных в этот регистр начинается передача, и подпрограмма ждет до тех пор, пока не будет передан набор символов.

```

SendeZeichen
    _BANK_1
    bsf TXSTA, TXEN      ;Разрешить передачу данных
    _BANK_0
    movwf TXREG
    _BANK_1
    btfss TXSTA, TRMT
    goto $-1             ;Ожидание завершения операции передачи данных
    _BANK_0
    return               ;Возврат из подпрограммы

```

Чтобы символы на экране ПК были отображены в определенном порядке, после передачи собственно символов должны передаваться еще и *управляющие символы*. Чтобы демонстрировать, какое воздействие имеется у управляющих символов, различные управляющие символы используются в зависимости от нажатия кнопок. Так, при нажатии на кнопку 4 персональный компьютер дополнительно выводит еще и звуковой сигнал. К самым важным управляющим символам относятся следующие два:

- "\n" — LF (Line Feed) — последующие символы будут выводиться в новой строке, разумеется, не в начале строки, а в месте, в котором представлялся последний символ;
- "\r" — CR (Carriage Return) — последующие символы будут выводиться в начале текущей строки.


```

taster3                                ;Если была нажата кнопка 3,
    movlw A'S'                         ;передаются символы "S3" и управляющие
    call SendeZeichen                 ;символы, соответствующие клавише <Enter>
    movlw A'3'
    call SendeZeichen
    movlw 0x0A                        ;Установить маркер на новую строку
    call SendeZeichen                 ;=управляющий символ LF (Line Feed)
    movlw 0x0D                        ;Установить маркер на начало строки
    call SendeZeichen                 ;=управляющий символ CR (Carriage Return)
    btfss TASTER_3                    ;Ожидание до размыкания кнопки 3
    goto $-1
    _DELAY_TMR1_US d'20000'           ;Ожидание 20 мс для исключениядребезга
                                        ;контактов кнопки
    goto main                         ;Перейти на основной цикл программы
taster4                                ;Если была нажата кнопка 4,
    movlw A'S'                         ;передаются символы "S4" и управляющие
    call SendeZeichen                 ;символы, соответствующие клавише <Enter>
    movlw A'4'                        ;дополнительно выводится сигнал
    call SendeZeichen                 ;персонального компьютера
    movlw A' '                         ;Передача пробела
    call SendeZeichen
    movlw 0x07                        ;Персональный компьютер выдает сигнал
    call SendeZeichen
    btfss TASTER_4                    ;Ожидание до размыкания кнопки 4
    goto $-1                           ;
    _DELAY_TMR1_US d'20000'           ;Ожидание 20 мс для исключениядребезга
                                        ;контактов кнопки
    goto main                         ;Перейти на основной цикл программы

```

После передачи строки символов программа возвращается снова в основной цикл, начиная с проверки принятия данных и нажатий на кнопки.

13. Передача данных по шине I²C

Сокращение *I²C* (Inter-Integrated Circuit) означает взаимодействие между интегральными схемами. Эта шина используется для сопряжения различных интегральных микросхем (ИС). Передача данных происходит лишь по двум линиям: линии данных SDA (Serial Data) и линии синхронизации SCL (Serial Clock). Иногда для двухпроводного интерфейса встречается также обозначение *TWI* (Two-Wire-Interface). Интерфейс был разработан компанией Philips Semiconductors (теперь NXP Semiconductors) и, получив большое распространение, поддерживается многими интегральными схемами, которые изготавливаются разными производителями. Данные передаются байтами последовательно со скоростью до 3,4 Мбит/с. Стандартная скорость интерфейса I²C составляет около 100 Кбит/с. Этой скорости, как правило, достаточно для передачи при небольшом количестве управляющих регистров в микросхеме. Поэтому интерфейс очень часто используется, чтобы записывать в регистры модулей соответствующие данные. Так как возможности интерфейса I²C очень обширные и разносторонние, то в этой книге затронут только принцип и простая передача данных через интерфейс. Подробную спецификацию на шину I²C можно найти на веб-странице компании NXP Semiconductors или на прилагаемом к книге компакт-диске.

13.1. Принцип работы интерфейса I²C

Каждая интегральная схема, которая присоединяется к шине I²C, имеет однозначный уникальный адрес, или точнее, *адрес устройства* (Device Address). Как правило, микроконтроллер является ведущим устройством шины, которое управляет обменом. *Ведущее устройство* (Master) посредством адреса обращается к присоединенным интегральным микросхемам — *ведомым устройствам* (Slave) или иначе подчиненным устройствам. После стартовой последовательности ведущее устройство передает адрес ведомого устройства и сообщает ему, что оно должно выполнять, передавать или читать данные. Это

указание направления передачи данных осуществляется посредством последнего бита чтения/записи в переданном байте адреса. При установке этого бита на высокий логический уровень данные должны считываться из ведомого устройства (R). Если же данные должны передаваться на ведомое устройство, то бит устанавливается на низкий уровень (\bar{W}). Когда ведомое устройство, декодировав адрес, определяет, что было обращение именно к нему, то оно подтверждает правильное получение адреса, передавая следующий бит подтверждения ACK (Acknowledge) низкого логического уровня (ACK = 0). Затем в большинстве случаев в микросхему передается внутренний адрес того или иного регистра. После этого передаются данные, которые должны быть записаны в указанный регистр, или наоборот считываются из этого регистра. На рис. 13.1 представлен пример последовательности передачи.

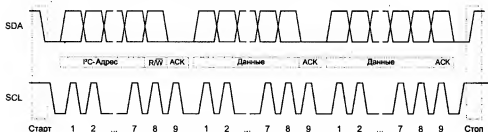


Рис. 13.1. Последовательность передачи по шине I²C

Поскольку имеется вероятность наличия нескольких ведущих устройств на шине, то может возникнуть опасность в том, что два цифровых выхода с противоположными сигналами будут мешать друг другу и могут даже привести к короткому замыканию. Чтобы обойти эту проблему, микросхемы должны иметь *выходы с открытым коллектором* (Open Collector). К выходам обязательно должны быть подключены резисторы, подключенные к напряжению высокого логического уровня, которые называют *подтягивающими резисторами*. Это значит, что в случае сигнала низкого логического уровня открытый выходной транзистор, соединенный с общей шиной, будет подтягивать выходную линию к нулевому потенциалу. Для получения сигнала высокого логического уровня выходной транзистор должен быть закрыт. В этом случае при одновременном открытии двух выходных транзисторов ток разделяется на оба транзистора и никакого короткого замыкания не возникает. С помощью внутренней аппаратной логики ведущее устройство может определить состояние шины в данный момент, и не будет передавать данные до тех пор, пока шина не будет свободна. Максимальная величина подтягивающего резистора зависит от емкости шины. Когда соединяются несколько микросхем или линии шины продолжительны, емкость шины возрастает, и поэтому

сопротивление подтягивающего резистора должно быть меньше, чтобы эти емкости смогли перезаряжаться достаточно быстро. Этот эффект критичнее при более высоких скоростях передачи данных, чем при стандартной скорости. Для стандартной скорости 100 Кбит/с сопротивление может быть в диапазоне между 3 и 10 кОм. Для более высоких скоростей передачи сопротивление должно выбираться между 1 и 5 кОм.

Чтобы присоединенные микросхемы могли определять, когда начинается и кончается передача, имеются специальные условия старта и стопа. Эти условия формирует ведущее устройство. Для начала передачи сигнал на линии данных SDA переводится из высокого на низкий логический уровень при высоком уровне сигнала синхронизации SCL. После формирования условия старта ("Старт") ведущим устройством начинается формирование импульсов синхронизации для передачи данных. По завершению передачи сигнал на шине синхронизации SCL переходит на высокий логический уровень, а затем сигнал на линии данных SDA переключается с низкого на высокий логический уровень (условие "Стоп"). Далее шина снова становится пассивной и может использоваться другими ведущими устройствами шины.

Чтобы гарантировать надежную передачу данных, изменение данных допускается только тогда, когда шина синхронизации находится на низком логическом уровне. Данные считаются действительными лишь во время высокого логического уровня сигнала синхронизации SCL.

Для организации обмена по шине I²C используется модуль MSSP (Master Synchronous Serial Port) PIC-микроконтроллера, и требуется только учитывать последовательность состояний сигнала. Большая часть работы выполняется внутренними аппаратными средствами микроконтроллера. Только для вывода условия старта или стопа в управляющих регистрах должен устанавливаться соответствующий бит.

13.2. Управление памятью EEPROM

Далее представляется пример, в котором имеется возможность сохранения данных во внешней EEPROM-памяти посредством шины I²C. В этом случае осуществляется опрос данных с аналогового входа, затем уже оцифрованные данные сохраняются в EEPROM-памяти. Возможное применение этого примера — запись температуры в течение длинного периода времени. После этого данные могут считываться посредством последовательного интерфейса и далее отображаться на персональном компьютере. Однако прежде чем начать с примера, сначала надо рассмотреть некоторые основы управления EEPROM-памятью.

Микросхемы EEPROM-памяти с поддержкой последовательного интерфейса доступны со стандартной емкостью запоминающего устройства от 128 битов

до 1 Мбит и поэтому скорее годятся только для небольших объемов данных. Если же требуется несколько больший объем памяти, то устанавливаются совместно несколько микросхем памяти и таким образом память может быть увеличена. Однако расширение ограничено количеством выводов для выбора микросхемы (Chip Select). Для примера в этой книге используется 32-килобитная EEPROM-память 24LC32A. К этой EEPROM-памяти обращаются с помощью управляющего байта, который состоит из 2 частей. Первые 4 бита являются управляющим кодом (Control Code), 3 следующих бита — это биты выбора микросхемы (Chip Select Bits), состояние которых определяется с помощью внешних выводов. Например, при наличии на выводе A2 высокого уровня и на обоих выводах A1 и A0 — низкого логического уровня, обращение к EEPROM-памяти будет осуществляться посредством адреса 1010100. Как можно видеть, здесь речь идет о 7 битах для адресации устройства, а восьмой бит используется для определения операции чтения или записи (R/W). Это приводит к небольшой проблеме в указании адреса устройства. Имеются производители, которые указывают 8-битный адрес для чтения и 8-битный адрес для записи (например, 0xA8 для записи и 0xA9 для чтения). С другой стороны, имеются производители, которые указывают адрес устройства как 7-битное значение (например, 0x54). На первый взгляд адреса выглядят разными, однако они одинаковые. Поэтому очень точно нужно проверять, как указан адрес в техническом описании микросхемы. Чтобы действовать наверняка, необходимо составлять адрес в двоичном коде и затем использовать его по своему желанию.

Во внутренней EEPROM-памяти данные сохраняются по байтам. Для обращения ко всем ячейкам памяти требуется 12-битный адрес ($2^{12} = 4096$ байт = = 32768 бит). Чтобы записать данные в EEPROM-память или считать из нее посредством интерфейса I²C, должен передаваться необходимый адрес. Для этого, в микросхему EEPROM-памяти передается сначала адрес устройства с указанием операции для выполнения (чтение/запись), затем посылаются 2 байта, в которых находится адрес ячейки памяти. Поскольку для указания адреса ячейки требуются только 12 битов, то старшие 4 бита адреса не представляют интереса и на них можно не обращать внимание.

Следует иметь в виду, что адрес конкретной ячейки памяти для любого читаемого или записываемого байта данных (слова данных) не надо передавать каждый раз. Можно передавать несколько данных в группе. В таком случае с каждым переданным байтом адрес ячейки внутри EEPROM-памяти автоматически увеличивается на единицу. Таким образом можно считать все данные из EEPROM-памяти. Кроме того, надо учитывать, что для записи за один раз может быть передано максимум 32 байта данных. Это связано с соответствующим размером имеющегося в EEPROM-памяти буфера, в котором временно сохраняются данные для последующей записи их в указанные ячейки

памяти. Этот буфер необходим в связи с тем, что процесс записи продолжается дольше, чем чтение.

На рис. 13.2 и 13.3 представлены примеры последовательности сигналов для записи данных в EEPROM-память. Последовательность сигналов для чтения данных приведена на рис. 13.4 и 13.5.

Перед чтением данных из EEPROM-памяти сначала нужно передавать адрес ячейки памяти, из которой должно выполняться чтение. Поэтому первоначально в управляющем байте после I²C-адреса нужно установить бит R/W на

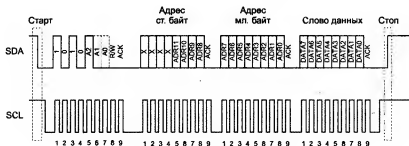


Рис. 13.2. Запись отдельного байта

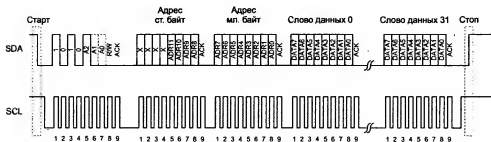


Рис. 13.3. Запись нескольких байтов

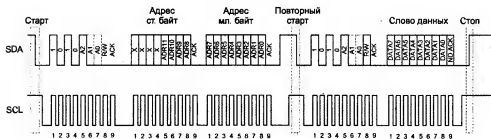


Рис. 13.4. Чтение отдельного байта

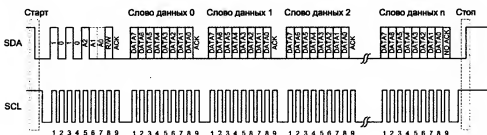


Рис. 13.5. Чтение нескольких байтов

низкий логический уровень, чтобы записать этот стартовый адрес ячейки в EEPROM-память. Вслед за этим повторно выполняется условие старта, чтобы сообщать после этого EEPROM-памяти при помощи I²C-адреса и бита R/\overline{W} , что теперь должна выполняться операция чтения.

13.3. Пример программы "Сохранение измеренных значений в EEPROM-памяти"

С помощью примера поясняется, как данные могут быть записаны в EEPROM-память посредством шины I²C. Чтобы получить данные для сохранения, с аналогового входа считывается напряжение и затем оцифровывается. Для управления операциями сохранения и чтения данных из памяти используются 2 кнопки. Чтобы считать данные из EEPROM-памяти, применяют кнопку 1 (Taster 1). После нажатия кнопки из EEPROM-памяти считываются 20 данных и передаются через последовательный интерфейс в персональный компьютер. Для сохранения данных в EEPROM-памяти требуется нажать на кнопку 2 (Taster 2). После этого аналоговое напряжение считывается, оцифровывается и пишется в память, начиная со следующего указанного ранее адреса ячейки памяти. Если в EEPROM-памяти сохраняются более 20 значений, то включается светодиод, который показывает условие переполнения. При каждом новом нажатии на кнопку данные перезаписываются с самого начала. Светодиод горит до тех пор, пока данные не будут считаны и память снова не освободится для поступления новых данных.

Чтобы использовать модуль MSSP для передачи данных, он соответствующим образом должен быть инициализирован для режима I²C. Имеются в целом 5 регистров, которые используются при передаче данных по интерфейсу I²C. С помощью регистра SSPADD можно установить скорость передачи данных.

Частота сигнала синхронизации SCL при обмене может рассчитываться по следующей формуле:

$$F = \frac{F_{osc}}{4 \cdot (SSPADD + 1)} \quad 13.1)$$

Из этой формулы для частоты передачи 100 кГц при тактовой частоте микроконтроллера $F_{osc} = 4$ МГц необходимо значение регистра SSPADD равно 9.

Для данных, которые должны передаваться или же приниматься, используется регистр SSPBUF. Кроме того, имеются еще 3 управляющих и контролируемых регистра, с помощью которых может запрашиваться или изменяться состояние процесса передачи. В регистре SSPSTAT отображается, например, определялся ли стартовый или стоповый бит, а также полон ли буфер SSPBUF при передаче или приеме данных. С помощью регистра SSPCON включается модуль MSSP и задается режим его работы, т.е. должен ли PIC-контроллер работать как ведущее или ведомое устройство в режиме I²C. Посредством регистра SSPCON2 могут выводиться различные комбинации сигналов SDA и SCL, соответствующие условиям: "Старт" (Start), "Стоп" (Stop) или подтверждение — сигнал ACK (Acknowledge).

Чтобы использовать интерфейс I²C для передачи данных в EEPROM-память, в примере выполняется следующая последовательность команд инициализации:

```
init_i2c                ;Инициализация модуля I2C
  _BANK_1
  movlw d'9'             ;Установить частоту синхронизации равную 100 кГц
  movwf SSPADD           ;F = Fosc/(4*(SSPADD+1))
  _BANK_0
  movlw b'00101000'      ;Включить модуль MSSP и конфигурировать PIC
  movwf SSPCON           ;как ведущее устройство в режиме I2C
  _BANK_1
  movlw b'10100000'      ;Настроить состояния модуля MSSP в режиме I2C
  movwf SSPSTAT          ;с помощью регистра SSPSTAT
  clrf SSPCON2           ;Очистить все биты в регистре SSPCON2
  _BANK_0
```

При выполнении инициализации сначала устанавливается скорость передачи данных равная 100 Кбит/с, путем записи значения 9 в регистр SSPADD. Затем с помощью битов SSPEN и SSPM3 регистра SSPCON включается модуль MSSP, а PIC-микроконтроллер конфигурируют, как ведущее устройство при обмене по интерфейсу I²C. В регистре SSPSTAT устанавливается бит SMP для управления скоростью изменения сигнала (Slew Rate Control) при стандартной скорости обмена. Все другие биты в регистре SSPSTAT очищаются. В конце инициализации в регистре SSPCON2 все биты очищаются.

Основная программа имеет относительно простую структуру и в основном выполняет определение той кнопки, которая была нажата, а после этого запускает соответствующую операцию.

```

main                                ;Начало основного цикла
    btfsc Taster_1                  ;Проверить, нажимали ли на кнопку 1
    goto prüfe_Taster2             ;Если нет, то перейти к проверке кнопки 2
    movlw a'\n'                     ;Чтобы символы представлялись на мониторе
    call SendeZeichen              ;в программе терминала (HyperTerminal) более
    movlw a'\r'                     ;наглядно, начать вывод в начале
    call SendeZeichen              ;новой строки
    movlw d'20'
    movwf COUNTER                  ;Задать 20 16-битных значений, которые будут
                                   ;читаться из EEPROM-памяти
    clrf ADR_L                     ;Задать начальный адрес ячейки памяти 0x0000
    clrf ADR_H                     ;для чтения
    call Lese_EEPROM              ;Приступить к чтению из EEPROM-памяти
                                   ;сохраненных данных
    bcf LED_1                      ;Погасить светодиод, если встретилось
                                   ;переполнение
    _DELAY_TMR1_US d'500000'      ;Ожидать 0,5 сек, поскольку иначе при более
                                   ;длительном нажатии кнопки содержимое EEPROM
                                   ;считывалось бы неоднократно
prüfe_Taster2                       ;Проверить, нажимали ли на кнопку 2
    btfsc TASTER_2                 ;Если кнопку 2 не нажимали,
    goto main                     ;перейти на начало программы
lese_AN0                           ;Если нажимали, то
    _BANK_1                       ;считать напряжение, поданное на вывод AN0
    clrf ADRESL                   ;Очистить регистр для хранения младшей
                                   ;части результата АЦП
    _BANK_0
    clrf ADRESH                   ;Очистить регистр для хранения старшей
                                   ;части результата АЦП
    bsf ADCON0, ADON              ;Включить модуль АЦП
    bsf ADCON0, GO_DONE           ;Запустить процедуру преобразования
    btfsc ADCON0, GO_DONE
    goto $-1                      ;Ожидать до тех пор, пока бит GO_DONE не
    пор                           ;будет сброшен аппаратно
    _BANK_1
    movf ADRESL, W                ;Скопировать оцифрованные напряжения в
    _BANK_0                       ;регистры DATEN_L и DATEN_H
    movwf DATEN_L
    movf ADRESH, W
    movwf DATEN_H

```

```

call Schreibe_EEPROM      ;Перейти к записи данных в EEPROM-память
clrfr ADR_H               ;Поскольку сохраняется только 20 16-битных
                           ;значений, то старший байт адреса
movlw d'2'                ;всегда должен быть 0x00
addwf ADR_L, F            ;Увеличить адрес на 2, поскольку
                           ;всегда сохраняются 16-битные значения
movlw d'40'               ;Проверить, сохранялись ли уже все
subwf ADR_L, W            ;20 значений (=40 байт)
btfsc STATUS, C
goto überlauf_EEPROM      ;Перейти на метку überlauf_EEPROM, если
                           ;уже было сохранено 20 значений
_DELAY_TMR1_US d'500000'  ;Ожидать 0,5 сек
goto main
überlauf_EEPROM
clrfr ADR_L               ;Если адрес больше чем 40,
                           ;начать снова с 0
bsf LED_1                 ;Включить LED_1 как предупреждение
_DELAY_TMR1_US d'500000'  ;Ожидать 0,5 сек
goto main

```

Вначале проверяется, было ли нажатие кнопки 1. Если было, то данные должны читаться из EEPROM-памяти и передаваться через последовательный интерфейс в компьютер для отображения их на мониторе, под управлением программы для работы на терминале, например, HyperTerminal. В этом случае, в компьютер передаются, прежде всего, 2 специальных символа "\n" (новая строка) и "\r" (начало строки), которые нужны для начала отображения данных с начала новой строки. Затем задается количество 16-битных значений, которые будут считываться из EEPROM-памяти. Поэтому количество, равное 20, записывается в регистр COUNTER. Затем задается начальный адрес 0x0000 ячейки памяти, с которого должно начинаться чтение. После вызова подпрограммы `Lese_EEPROM` данные считываются из EEPROM-памяти и передаются непосредственно через последовательный интерфейс. После того как будут считаны и переданы все данные, можно отключить светодиод, который сигнализирует о переполнении. Затем для исключения неоднократного последовательного считывания EEPROM-памяти при длительно нажатой кнопке задается небольшая временная задержка длительностью 0,5 секунд. После этого снова осуществляется переход на начало основной программы и вновь делается опрос состояния кнопок.

При нажатой кнопке 2 все значения для измеренного напряжения должны сохраняться в EEPROM-памяти. Напряжение поступает на аналоговый вход AN0. После нажатия на кнопку 2 напряжение на выводе AN0 сначала оцифровывается и далее копируется в регистры `DATEN_L` и `DATEN_H`. Затем после вызова подпрограммы `Schreibe_EEPROM` данные из обоих регистров сохраняются

в EEPROM-памяти. Адрес в регистрах `ADR_L` и `ADR_H` после каждой операции записи в память увеличивают на 2, поскольку в EEPROM-память всегда пишут 2 байта (16 битов). После увеличения адреса проверяется, достигнут ли уже максимальный адрес. Если это происходит, включается светодиод, который показывает переполнение EEPROM-памяти, и регистры для хранения адреса памяти снова сбрасываются на 0. Если же переполнения не было, то выполняется переход на начало программы для продолжения сохранения данных в EEPROM-памяти.

13.3.1. Подпрограмма *Schreibe_EEPROM*

Для сохранения данных в EEPROM-памяти используется подпрограмма *Schreibe_EEPROM*. В ней устанавливаются соответствующие управляющие биты и запрашиваются биты состояния.

Schreibe_EEPROM

```

_BANK_0
bcf PIR1, SSPIF           ;Сбросить флаг прерывания от модуля MSSP
_BANK_1
bcf SSPCON2, RCEN         ;Переключиться на режим передачи данных
bsf SSPCON2, SEN          ;Сгенерировать условие "Старт"
btfsc SSPCON2, SEN        ;Ожидать до тех пор, пока условие "Старт"
goto $-1                  ;не будет сгенерировано
_BANK_0
movlw EEPROM_ADR_S       ;Передать I2C-адрес EEPROM-памяти
movwf SSPBUF
bcf PIR1, SSPIF           ;Сбросить флаг прерывания от модуля MSSP
_BANK_1
btfsc SSPSTAT, R_W        ;Ожидать до тех пор, пока адрес
goto $-1                  ;не будет выведен
btfsc SSPCON2, ACKSTAT    ;Проверить бит подтверждения
goto $-1                  ;Ожидание подтверждения от ведомого устройства
_BANK_0
bcf PIR1, SSPIF           ;Сбросить флаг прерывания от модуля MSSP
movf ADR_H, W             ;Передать старший байт адреса ячейки памяти
movwf SSPBUF
_BANK_1
btfsc SSPSTAT, R_W        ;Ожидать до тех пор, пока не будет выведен
goto $-1                  ;старший байт адреса ячейки памяти
btfsc SSPCON2, ACKSTAT    ;Проверить бит подтверждения
goto $-1                  ;Ожидание подтверждения от ведомого устройства
_BANK_0
bcf PIR1, SSPIF           ;Сбросить флаг прерывания от модуля MSSP
movf ADR_L, W             ;Передать младший байт адреса ячейки памяти
movwf SSPBUF

```



```

_BANK_1
btfsc SSPSTAT, R_W      ;Ожидать до тех пор, пока не будет выведен
goto $-1                ;младший байт адреса ячейки памяти
btfsc SSPCON2, ACKSTAT   ;Проверить бит подтверждения
goto $-1                ;Ожидание подтверждения от ведомого устройства
_BANK_0
bcf PIR1, SSPIF          ;Сбросить флаг прерывания от модуля MSSP
movf DATEN_H, W          ;Передать старший байт данных
movwf SSPBUF
_BANK_1
btfsc SSPSTAT, R_W
goto $-1
btfsc SSPCON2, ACKSTAT   ;Проверить бит подтверждения
goto $-1                ;Ожидание подтверждения от ведомого устройства
_BANK_0
bcf PIR1, SSPIF          ;Сбросить флаг прерывания от модуля MSSP
movf DATEN_L, W          ;Передать младший байт данных
movwf SSPBUF
_BANK_1
btfsc SSPSTAT, R_W
goto $-1
btfsc SSPCON2, ACKSTAT   ;Проверить бит подтверждения
goto $-1                ;Ожидание подтверждения от ведомого устройства
_BANK_0
bcf PIR1, SSPIF          ;Сбросить флаг прерывания от модуля MSSP
_BANK_1
bsf SSPCON2, PEN         ;Сгенерировать условие "Стоп"
_BANK_0
btfss PIR1, SSPIF        ;Проверить, были ли прочитаны данные
goto $-1
bcf PIR1, SSPIF          ;Сбросить флаг прерывания от модуля MSSP
_BANK_0
btfss TASTER_2
goto $-1
nop
return

```

Вначале сбрасывается флаг прерывания от модуля MSSP, если он был установлен еще при предыдущей операции. Затем модуль MSSP переводится в режим передачи и генерируется условие "Старт" после установки бита SEN в регистре SSPCON2. Если условие "Старт" успешно выдано, то бит SEN аппаратными средствами будет автоматически сброшен. Запрашивая и проверяя этот бит, становится понятно, когда можно начать выдачу адреса EEPROM-памяти. Адрес устройства для записи данных (EEPROM_ADR_S) выводится авто-

матически, как только значение адреса будет записано в стандартный регистр SSPBUF. Вслед за этим устанавливается бит R/\overline{W} на высокий уровень и запускается вывод. После вывода всех 8 битов бит R/\overline{W} должен быть снова сброшен на низкий логический уровень. Можно устанавливать этот бит с помощью его проверки, когда данные будут полностью выведены. Вслед за этим определенное ведомое устройство должно еще подтвердить успешное получение адреса. Это происходит во время 9-го такта, когда ведомое устройство на линию данных SDA выдает сигнал низкого логического уровня. Это событие можно определить при запросе бита ACKSTAT в регистре SSPCON2. В том случае, когда адрес не будет принят ведомым устройством, линия SDA во время девятого такта остается на высоком логическом уровне, и программа зависнет на этом месте. Если же встретится такая проблема, то сначала нужно проверить, правильно ли задано значение сопротивления для подтягивающего резистора и правильно ли производились все установки (адрес, режим ведущего устройства и т. д.). Если получение адреса устройства было подтверждено ведомым устройством, то после этого можно начать передавать начальный адрес ячейки памяти, в которую будет выполняться запись. Выполнение этой передачи осуществляется аналогично передаче адреса устройства. Сначала передается старший байт адреса, затем младший. Каждый переданный байт квитируется битом подтверждения ACK (Acknowledge), получаемым от EEPROM-памяти. Вслед за этим передаются оба слова данных из регистров DATEN_L и DATEN_H. После передаче в EEPROM-память 16-битного значения оцифрованных данных и получения подтверждения на каждый переданный байт, больше пока никакие данные передаваться не должны. Поэтому в это время генерируется условие "Стоп" и таким образом шина I²C снова освобождается для других операций. Условие "Стоп" генерируется путем установки бита PEN в регистре SSPCON2. После того как условие "Стоп" будет выдано на шину, вызывается прерывание, и бит PEN сбрасывается. В это время как раз и проверяется установка флага прерывания модулем MSSP. Иначе можно было бы запрашивать и проверять бит PEN. Теперь передача закончена и обе линии SDA и SCL с помощью подтягивающих резисторов переводятся на высокий логический уровень. В самом конце подпрограммы проверяется, отпущена ли кнопка 2, чтобы данные, при длительно нажатой кнопке, не записывались снова в EEPROM-память. Выход из подпрограммы будет только после размыкания контактов этой кнопки.

13.3.2. Подпрограмма *Lese_EEPROM*

В подпрограмме *Lese_EEPROM* данные считываются из EEPROM-памяти и передаются через последовательный интерфейс.

Lese_EEPROM

_BANK_0

bcf PIR1, SSPIF

;Сбросить флаг прерывания от модуля MSSP

```

    _BANK_1
    bsf SSPCON2, SEN      ;Сгенерировать условие "Старт"
    btfsc SSPCON2, SEN    ;Ожидать до тех пор, пока условие "Старт"
    goto $-1              ;не будет сгенерировано
    _BANK_0
    bcf PIR1, SSPIF       ;Сбросить флаг прерывания от модуля MSSP
    movlw EEPROM_ADDR_S   ;Передать I2C-адрес EEPROM-памяти
    movwf SSPBUF
    _BANK_1
    btfsc SSPSTAT, R_W    ;Ожидать до тех пор, пока адрес
    goto $-1              ;не будет выведен
    _BANK_0
    movf ADR_H, W         ;Передать старший байт адреса ячейки памяти
    movwf SSPBUF
    _BANK_1
    btfsc SSPSTAT, R_W    ;Ожидать до тех пор, пока не будет выведен
    goto $-1              ;старший байт адреса ячейки памяти
    _BANK_0
    movf ADR_L, W         ;Передать младший байт адреса ячейки памяти
    movwf SSPBUF
    _BANK_1
    btfsc SSPSTAT, R_W    ;Ожидать до тех пор, пока не будет выведен
    goto $-1              ;младший байт адреса ячейки памяти
    bsf SSPCON2, RSEN     ;Повторное генерирование условия "Старт"
    btfsc SSPCON2, RSEN
    goto $-1
    _BANK_0
    bcf PIR1, SSPIF       ;Сбросить флаг прерывания от модуля MSSP
    movlw EEPROM_ADDR_L   ;Передать адрес для чтения из EEPROM-памяти
    movwf SSPBUF
    _BANK_1
    btfsc SSPSTAT, R_W    ;Ожидать до тех пор, пока адрес
    goto $-1              ;не будет выведен
loop_read
    movlw a'0'            ;Вывод префикса "0x" для шестнадцатеричных чисел
    call SendeZeichen
    movlw a'x'
    call SendeZeichen
    ;Чтение старшего байта
    _BANK_0
    bcf PIR1, SSPIF       ;Сбросить флаг прерывания от модуля MSSP
    _BANK_1
    bsf SSPCON2, RCEN     ;Переключиться в режим приема

```

```

_BANK_0
btfss PIR1, SSPIF      ;Проверить, считались ли данные
goto $-1
bcf PIR1, SSPIF        ;Сбросить флаг прерывания от модуля MSSP
_BANK_1
bcf SSPCON2, ACKDT     ;Установить бит подтверждения
                        ; (Acknowledge)
bsf SSPCON2, ACKEN     ;Активизировать условие подтверждения
                        ; (вывод сигнала ACK)
btfsc SSPCON2, ACKEN   ;Ожидать до тех пор, пока сигнал подтверждения
goto $-1               ; не будет выведен
_BANK_0
bcf PIR1, SSPIF        ;Сбросить флаг прерывания от модуля MSSP
movf SSPBUF, W         ;Получить данные из буфера I2C
movwf DATEN_H
call Bin2Ascii         ;Преобразовать двоичное значение
                        ; в ASCII-значение
movf HEX_H, W          ;Передать через последовательный интерфейс
call SendeZeichen      ;старшие 4 бита ASCII-значения
movf HEX_L, W          ;Передать через последовательный интерфейс
call SendeZeichen      ;младшие 4 бита ASCII-значения
;Чтение младшего байта
_BANK_0
bcf PIR1, SSPIF        ;Сбросить флаг прерывания от модуля MSSP
_BANK_1
bsf SSPCON2, RCEN      ;Переключиться в режим приема
_BANK_0
btfss PIR1, SSPIF      ;Проверить, считались ли данные
goto $-1
bcf PIR1, SSPIF        ;Сбросить флаг прерывания от модуля MSSP
_BANK_1
bcf SSPCON2, ACKDT     ;Установить бит подтверждения
                        ; (Acknowledge)
bsf SSPCON2, ACKEN     ;Активизировать условие подтверждения
                        ; (вывод сигнала ACK)
btfsc SSPCON2, ACKEN   ;Ожидать до тех пор, пока сигнал ACK
goto $-1               ; не будет выведен
_BANK_0
bcf PIR1, SSPIF        ;Сбросить флаг прерывания от модуля MSSP
movf SSPBUF, W         ;Получить данные из буфера I2C
movwf DATEN_L
call Bin2Ascii         ;Преобразование двоичного значения
                        ; в ASCII-значение

```

```

movf HEX_H, W           ;Передать через последовательный интерфейс
call SendeZeichen        ;старшие 4 бита ASCII-значения
movf HEX_L, W           ;Передать через последовательный интерфейс
call SendeZeichen        ;младшие 4 бита ASCII-значения
movlw a'\n'             ;Перейти на новую строку
call SendeZeichen
movlw a'\r'             ;в начало строки
call SendeZeichen
decfsz COUNTER           ;Уменьшить на 1 содержимое счетчика (COUNTER)
goto loop_read          ;Если в счетчике 0, то никакие
                        ;значения больше не должны выводиться
                        ;и передача может быть закончена

_BANK_1
bsf SSPCON2, ACKDT       ;Установить бит подтверждения данных
                        ;(нет передачи подтверждения)

bsf SSPCON2, ACKEN       ;Активизировать выдачу условия подтверждения
                        ;(вывод сигнала ACK)

btfsc SSPCON2, ACKEN     ;Ожидать до тех пор, пока сигнал подтверждения
goto $-1                ;не будет выведен
bcf SSPCON2, RCEN        ;Переключиться на режим передачи

_BANK_1
bsf SSPCON2, PEN         ;Сгенерировать условие "Стоп"
btfsc SSPSTAT, R_W
goto $-1                ;Ожидать до тех пор, пока условие "Стоп"
                        ;не будет выведено

_BANK_0
btfss TASTER_1           ;Ожидать до тех пор, пока кнопка 1
goto $-1                ;не будет отпущена
return

```

Выполнение операции чтения данных с передачей их по шине I²C функционирует аналогично записи. Сначала выводится адрес устройства для его записи. Это выглядит несколько запутанным на первый взгляд, но сначала требуется сообщить EEPROM-памяти, с какого адреса ячейки памяти должно выполняться чтение. Поэтому, прежде всего, в EEPROM-память передают значение старшего, а затем младшего байта адреса памяти из соответствующих регистров `ADR_H` и `ADR_L`. После выдачи адреса EEPROM-память находится в режиме чтения данных и далее должна переходить в режим передачи (записи), чтобы передавать данные из ячеек памяти. Для этого выполняется повторное генерирование условия "Старт". Последовательность действий та же самая, как и при обычном условии "Старт", разумеется, до сего времени никакого условия "Стоп" при этом не генерируется. После выдачи повторного условия "Старт" могут передаваться адреса устройств для чтения

(EEPROM_ADR_L). Однако прежде чем данные будут читаться через последовательный интерфейс, вначале передается символы префикса "0x" для обозначения шестнадцатеричных данных, чтобы он стоял перед каждым значением. Затем PIC-контроллер переключается в режим приема путем установки бита RCEN в регистре SSPCON2. После установки этого бита данные будут передаваться из EEPROM-памяти в PIC-контроллер. В конце передачи генерируется прерывание, которое может оцениваться с помощью бита прерывания SSPIF от модуля MSSP. Теперь после успешного получения PIC-контроллер должен подтвердить EEPROM-памяти получение данных посредством передачи сигнала подтверждения ACK. Для этого бит ACKDT в регистре SSPCON2 сбрасывается на 0. Затем после установки бита ACKEN активизируется условие выдачи подтверждения. После этого поступившие данные можно получить из регистра SSPBUF. Так как сначала сохранялся старший байт данных, то первое слово данных копируется в регистр DATEN_N. 8-битное слово данных сохраняется в регистре DATEN_N как цифровое значение между 0 (0x00) и 255 (0xFF). Чтобы это значение при отображении на мониторе с помощью терминальной программы было предельно понятно, оно должно преобразовываться в ASCII-формат. Для этого вызывается подпрограмма Bin2Ascii. В подпрограмме 8-битное значение разделяется на два 4-битных значения и анализируется каждая часть. В этом случае проверяется, чему соответствует 4-битное значение: цифре (0—9) или букве (A—F). Если соответствует цифре, то к коду числа добавляется 0x30, чтобы получить цифру в ASCII-формате, а если соответствует букве, то добавляют 0x37. Полная подпрограмма находится в примере на прилагаемом к книге компакт-диске. После вызова подпрограммы значение старшего полубайта в ASCII-коде будет в регистре HEX_N, а значение младшего полубайта в регистре HEX_L. Теперь эти оба регистра передаются с помощью последовательного интерфейса по очереди и отображаются за ранее переданным префиксом. Вслед за этим таким же способом читается следующее значение из EEPROM-памяти и затем также передается через последовательный интерфейс. На мониторе при этом 16-битное значение будет отображаться в виде "0xABCD". Кроме этого, при каждом выводе данных для начала их отображения с первой позиции в новой строке по последовательному интерфейсу передаются еще два специальных символа "\n" и "\r". Таким образом, первое 16-битное значение будет успешно передано, и счетчик может быть уменьшен на 1. Если в регистре счетчика COUNTER будет значение большее 0, то далее читается следующее слово данных и цикл повторяется до тех пор, пока все данные не будут переданы. После того как последние данные будут считаны, бит подтверждения не сбрасывается в состояние логического 0, при этом уровень сигнала подтверждения имеет высокий уровень. Таким способом EEPROM-память определяет, что передача данных закончена. После этого PIC-контроллер снова пере-

ключается в режим передачи и генерируется условие "Стоп". Теперь шина I²C снова будет свободна и предоставлена в распоряжение другим устройствам, подключенным к шине. В самом конце подпрограмма анализирует состояние кнопки 1 и ожидает момента ее отпускания, и только потом возвращается на выполнение основной программы.

Представленный пример очень прост и может быть по желанию усовершенствован. Например, вместо использования кнопок для управления, можно применить специальные команды, передаваемые по последовательному интерфейсу. Кроме того, можно устанавливать временной цикл и таким образом запрашивать аналоговое напряжение циклически. При этом если к аналоговому входу присоединить датчик температуры или влажности, то в течение более длинного периода можно будет наблюдать, как изменяются температура и влажность в помещении. Данные, которые будут собраны, например, в течение одной недели, могут затем быть обработаны на персональном компьютере.

14. Переключение с помощью инфракрасного дистанционного управления

При помощи *инфракрасного (ИК) дистанционного управления* осуществляется управление почти каждым телевизором и спутниковым приемником. Переключение программ в настоящее время уже практически не возможно без использования дистанционного пульта управления. Такой пульт может иметь до 30 кнопок, а при управлении более простым устройством это количество кнопок, разумеется, можно сэкономить. Для демонстрации возможности использования дистанционного управления в этой главе представлен пример, с помощью которого можно рассмотреть работу *инфракрасного протокола* (англ. *IR-Protocol* — Infrared Protocol). В данном примере светодиод переключается в зависимости от нажатия на кнопку. Если же вместо светодиода подключить реле, то можно было бы коммутировать нагрузку с напряжением сети 220 В (например, лампу). Таким способом можно относительно просто дистанционно управлять осветительной лампой. Для экономии электроэнергии можно установить подобную схему перед колодкой соединительных розеток и таким образом, нажатием кнопок на пульте дистанционного управления, отключать от сети несколько нагрузок (например, телевизор, приемник, цифровой плеер и т. д.). Вследствие этого присоединенные устройства больше не будут потреблять ток и расходы на электроэнергию сократятся.

К сожалению, производители инфракрасного (ИК) управления так и не смогли договориться о создании унифицированного стандарта для передачи. Поэтому теперь имеется много различных кодов, с помощью которых осуществляется дистанционное управление устройствами. В связи с этим в данной книге используется код, который относительно широко распространен и охотно применяется радиолюбителями для управления в собственных разработках. Имеется в виду *RC5-код*, который был разработан фирмой Philips. Если в библиотеке устройства телеуправления отсутствует управление, использующее код RC5, то имеется возможность приобрести универсальное

устройство дистанционного управления примерно за 10 €, при помощи которого можно передавать почти любой код. Код в устройстве программируется, как правило, с помощью трехзначной цифры. Чтобы найти нужный код, можно загрузить в микроконтроллер готовый пример программы. В этом случае на универсальном устройстве дистанционного управления можно запускать автоматический поиск. Когда будет передаваться именно код RC5, то светодиод будет переключаться. В дальнейшем можно будет использовать именно этот код для других разработок.

14.1. Протокол RC5

Чтобы гарантировать по возможности более стабильную передачу, при использовании протокола RC5 передают не просто высокий и низкий уровень сигнала, а для каждого бита посылается переменный сигнал. Кроме того, инфракрасный сигнал еще модулируется с частотой 36 кГц для защиты от помех, которые могут возникнуть от внешней засветки. При разработке было учтено, чтобы передача происходила по возможности с большей экономией энергии. Для повышения надежности код нажатой кнопки посылается три раза последовательно при каждом нажатии кнопки.

Для достижения 32-импульсной модуляции модуляция выполняется с частотой 36 кГц. Длительность периода импульса составляет 27,777 мкс ($T = 1 / f = 1 / 36 \text{ кГц}$). Во время передачи инфракрасный диод включен в течение 6,944 мкс ($1/4$ длительности периода) (рис. 14.1).

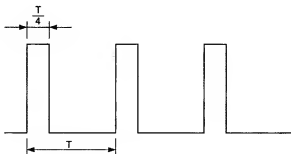


Рис. 14.1. Соотношение пауза-импульс инфракрасного передатчика

При передаче логической 1 ничего не передается в течение 32 периодов (рис. 14.2). Затем передаются 32 импульса с частотой 36 кГц. Для логического 0 передача происходит в обратном порядке. Сначала посылается пачка импульсов из 32 импульсов, а вслед за этим следует пауза длительностью равной 32 импульсам.

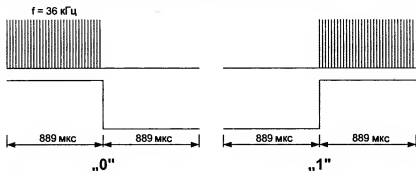


Рис. 14.2. Передача логических сигналов с модуляцией импульсами с частотой 36 кГц

Поэтому передача логического 0 или логической 1 продолжается 1,777 мс (64/36 кГц). Очень хорошая помехоустойчивость обусловлена за счет изменения уровня сигнала в течение передачи 1 бита. Сигнал должен обязательно изменить свой уровень после 1,777 мс. Если это не происходит, то возникает ошибка или же выполняется передача посредством другого протокола. Поэтому для безошибочного определения посланного передатчиком сигнала приемник должен быть обязательно синхронизирован.

В данном примере на стороне получения сигнала применяется приемник производства компании Vishay. Имеется в виду инфракрасный интегральный фотоприемник типа TSOP1736. В нем выполняется первая ступень обработки сигнала. Происходит регулирование усиления и сигнал фильтруется с помощью полосового фильтра с центральной частотой 36 кГц. Сигнал демодулируется и подается на выход через выходной транзистор. Транзистор присоединяется через внутренний подтягивающий резистор к напряжению питания. Поэтому в пассивном состоянии на выходе получается высокий логический уровень сигнала. Это означает, что на выходе приемника будет выводиться инверсный сигнал инфракрасного протокола. Для лучшего понимания приведенной далее программы уровни сигнала представлены в дальнейшем так, как они поступают на выходе модуля приемника. Поэтому для представления логической 1 требуется изменение уровня от высокого к низкому, а при передаче логического 0 изменение уровня сигнала от низкого к высокому уровню.

Код нажатой кнопки имеет длительность 24,889 мс. Это соответствует 14 битам с длительностью 1,777 мс. В эти 14 передаваемых бита входят 2 стартовых бита, 1 бит переключения, 5 битов адреса и 6 битов команды (рис. 14.3). Стартовые биты S_0 и S_1 используются для определения запуска инфракрасного протокола. Вслед за этим передается 1 бит переключения T_0 (Toggle Bit), с помощью которого можно определить, была ли нажата кнопка несколько раз подряд или она была нажата длительно. Этот бит переключения меняется после каждого нажатия кнопки с 0 на 1 или с 1 на 0. При дли-

тельном нажатии на кнопку бит переключения всегда сохраняет одно и то же значение. После бита переключения передаются 5 разрядов адреса A_4 — A_0 , которые определяют, в какое устройство выполняется передача. В табл. 14.1 представлены адреса различных устройств, используемых для инфракрасного управления.

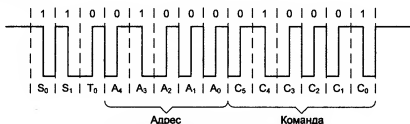


Рис. 14.3. Передача кода нажатой кнопки в инфракрасном дистанционном управлении

Таблица 14.1. Адреса для инфракрасной передачи

Адрес	Устройство	Адрес	Устройство
0x00	TV 1	0x10	Предусилитель 1
0x01	TV 2	0x11	Тюнер
0x02	Видеотекст	0x12	Рекордер 1
0x03	Видео VD	0x13	Предусилитель 2
0x04	Видео LV 1	0x14	CD-ROM
0x05	Видеомагнитофон 1	0x15	Телефон
0x06	Видеомагнитофон 2	0x16	Спутниковый приемник
0x07	Экспериментальный	0x17	Цифровой аудиоманитофон
0x08	Спутниковый приемник 1	0x18	—
0x09	Фотокамера	0x19	—
0x0A	Спутниковый приемник 2	0x1A	Перезаписываемый компакт-диск
0x0B	—	0x1B	—
0x0C	Компакт-диск с видеoinформацией	0x1C	—
0x0D	Видеокамера	0x1D	Освещение 1
0x0E	—	0x1E	Освещение 2
0x0F	—	0x1F	Телефон

После адреса устройства передаются 6 битов соответствующей команды. С помощью этих 6 битов могут передаваться 64 различных кода для нажимаемых кнопок на пульте дистанционного управления. Самые важные команды, которыми располагает почти каждое дистанционное управление, представлены в табл. 14.2.

Таблица 14.2. Команды для инфракрасной передачи

Команда	Кнопка	Команда	Кнопка	Команда	Кнопка	Команда	Кнопка
0x00	0	0x10	Громкость +	0x20	Канал +	0x30	Пауза
0x01	1	0x11	Громкость –	0x21	Канал –	0x31	
0x02	2	0x12	Яркость +	0x22		0x32	Быстро назад
0x03	3	0x13	Яркость –	0x23		0x33	—
0x04	4	0x14	Цвет +	0x24		0x34	Быстро вперед
0x05	5	0x15	Цвет –	0x25		0x35	Воспроизведение
0x06	6	0x16	НЧ +	0x26		0x36	Стоп
0x07	7	0x17	НЧ –	0x27		0x37	Прием
0x08	8	0x18	ВЧ +	0x28		0x38	
0x09	9	0x19	ВЧ –	0x29		0x39	
0x0A		0x1A	Баланс вправо	0x2A		0x3A	
0x0B		0x1B	Баланс влево	0x2B		0x3B	
0x0C	Режим ожидания	0x1C		0x2C		0x3C	
0x0D	Отключение звука	0x1D		0x2D		0x3D	
0x0E		0x1E		0x2E		0x3E	
0x0F		0x1F		0x2F		0x3F	

Итак, после нажатия любой кнопки управления будет выполняться передача, показанная на рис. 14.3. Сначала передаются оба стартовых бита и далее бит переключения. В качестве адреса в данном случае был выбран спутниковый приемник (адрес 0x08) и передана команда "Громкость –" (уменьшение гром-

протокол RC5. В данном примере светодиоды переключаются в зависимости от нажатой кнопки. Светодиод LED 4 всегда светится и выключается только, если определен допустимый действующий код. При нажатии кнопки "Громкость +" включается светодиод LED 1 и светится одну секунду, затем происходит переключение снова на LED 4. Если была нажата кнопка "Громкость -", то микроконтроллер выключает светодиод LED 4 и включает LED 2. Если нажимают на какую-либо другую кнопку, светится светодиод LED 3 примерно в течение одной секунды.

Также на прилагаемом к книге компакт-диске находятся данные для симуляции. Вследствие этого есть возможность выполнения программы в пошаговом режиме. На практике это, к сожалению, не так-то просто. Передача кода не может прерываться, т. к. всегда посылается полный код нажатой кнопки. Кроме того, на практике могут возникнуть и другие проблемы. Может случиться так, что универсальное устройство дистанционного управления не точно соблюдает синхронизацию по стандарту RC5. Вследствие этого для каждого бита появляются незначительные задержки, которые складываются в течение передачи. Если это происходит, то в конце протокола выборка информационного сигнала будет проходить не в середине бита, а поблизости от фронтов импульса. Тогда при слишком больших различиях возникают ошибки при передаче. Если это случается, то в подпрограмме должны применяться временные задержки. К сожалению, тогда для точного анализа требуется осциллограф, который имеется в распоряжении далеко не каждого радиолубителя.

В основной программе ИК-приемник длительно опрашивается в цикле. Если уровень изменяется с высокого на низкий, то начинается анализ протокола.

```
main                ;Начало основного цикла
    bcf LED_1        ;Выключить LED 1
    bcf LED_2        ;Выключить LED 2
    bcf LED_3        ;Выключить LED 3
    bsf LED_4        ;Включить LED 4, если никакой код не принимался
                    ;или же возникла ошибка
    btfsf IR_RCV     ;Проверить, изменился ли сигнал
                    ;ИК-приемника
    goto main        ;Никакое изменение сигнала не определено
```

После того как основная программа определила, что было изменение, далее проверяются оба стартовых бита. Таким способом проверяется, используется ли в данном случае код RC5. Если синхронизация нарушена уже на этом месте, то обработка принятых данных прерывается.

```

startbits_prüfen
    movlw d'14'           ;Задать в счетчике IR_COUNTER количество битов
    movwf IR_COUNTER      ;для считывания кода нажатой кнопки, равное 14
    bsf IR_ABTA$T         ;Указать момент выборки
    _DELAY_TMR1_US d'445' ;Временная задержка, чтобы делать выборку
                           ;в середине следующего сигнала
    bcf IR_ABTA$T         ;Указать момент выборки
    btfsc IR_RCV           ;Первый стартовый бит определен
    goto fehler           ;с ошибкой
    decf IR_COUNTER       ;Первый стартовый бит распознан и допустим
    btfsc STATUS, Z       ;Проверить, сосчитал ли счетчик IR_COUNTER
                           ;до 0, считая в обратном порядке
    goto fehler           ;Счетчик сосчитал до 0
    _DELAY_TMR1_US d'889'
    bsf IR_ABTA$T         ;Указать момент выборки
    btfss IR_RCV          ;Первая половина 2-го стартового бита
                           ;должна быть высокого уровня
    goto fehler           ;Ошибка
    _DELAY_TMR1_US d'889'
    bsf IR_ABTA$T         ;Указать момент выборки
    btfsc IR_RCV          ;Вторая половина 2-го стартового бита
                           ;должна быть низкого уровня
    goto fehler           ;Ошибка
    decf IR_COUNTER       ;Уменьшить содержимое счетчика IR_COUNTER на 1
    btfsc STATUS, Z       ;Проверить, досчитал ли счетчик
                           ;до 0, считая в обратном порядке
    goto fehler           ;Счетчик сосчитал до 0

```

Во время обработки поступающих данных выход RC5 микроконтроллера переключается перед каждым моментом выборки данных с ИК-приемника. Если посмотреть этот исходный уровень при симуляции или измерить его при помощи осциллографа, то на этом выводе можно увидеть, в какой момент осуществляется выборка ИК-сигнала. Благодаря этому можно легко определить сбой в синхронизации. С помощью регистра IR_COUNTER считается количество уже обработанных битов. Если до окончания анализа этот счетчик, считая в обратном направлении, досчитал до 0, то в данном случае произошла ошибка и осуществляется переход на метку fehler.

Так как проверка отдельных битов повторяется и протекает всегда одинаково, то для этой цели была написана специальная подпрограмма IR_Bittest.

```

IR_Bittest
    clrf IR_FLANKE       ;Очистить старые данные из регистра
                           ;IR_FLANKE

```



```

    clrf IR_STATUS      ;Очистить старые данные из регистра
                        ;IR_STATUS

ir_test_1              ;Проверка первой половины бита
    _DELAY_TMR1_US d'875' ;Временная задержка – меньше чем 889 мкс,
                        ;т. к. для команд также
                        ;требуется время для
    bsf IR_ABTASt       ;определения места расположения
                        ;точек выборки
    btfsc IR_RCV         ;Проверить уровень сигнала
                        ;ИК-приемника
    goto ir_high_1       ;Первая половина бита высокого уровня
    goto ir_low_1        ;Первая половина бита низкого уровня
ir_high_1              ;
    bsf IR_FLANKE, 1     ;При высоком уровне сигнала установить бит 1
                        ;в регистре IR_FLANKE
    goto ir_test_2       ;Проверить вторую половину бита
ir_low_1               ;
    bcf IR_FLANKE, 1     ;При низком уровне сигнала очистить бит 1
                        ;в регистре IR_FLANKE
    goto ir_test_2       ;Проверить вторую половину бита
ir_test_2              ;Проверка второй половины бита
    _DELAY_TMR1_US d'860' ;Временная задержка – менее 889 мкс,
                        ;т. к. для команд также
                        ;требуется время для
    bcf IR_ABTASt       ;определения места расположения
                        ;точек выборки
    btfsc IR_RCV         ;Проверить уровень сигнала
                        ;ИК-приемника
    goto ir_high_2       ;Вторая половина бита высокого уровня
    goto ir_low_2        ;Вторая половина бита низкого уровня
ir_high_2              ;
    bsf IR_FLANKE, 0     ;При высоком уровне установить бит 0
                        ;в регистре IR_FLANKE
    goto ir_test_ende    ;Обе половины бита
                        ;проверены -> завершить проверку
ir_low_2               ;
    bcf IR_FLANKE, 0     ;При низком уровне очистить бит 0
                        ;в регистре IR_FLANKE
    goto ir_test_ende    ;Обе половины бита
                        ;проверены -> завершить проверку
ir_test_ende           ;
    nop                  ;Окончание проверки состояния ИК-сигнала

```

```

ir_bittest_low
    movlw 0x01                ;Проверка логического 0
    subwf IR_FLANKE, W
    btfss STATUS, Z
    goto ir_bittest_high
    bsf IR_STATUS, 0          ;Если бит равен 0, то установить бит 0
    bcf IR_STATUS, 1          ;в регистре IR_STATUS,
                                ;а бит 1 очистить
    bcf IR_STATUS, 2          ;Очистить бит ошибки
    return

ir_bittest_high
    movlw 0x02                ;Проверка логической 1
    subwf IR_FLANKE, W
    btfss STATUS, Z
    goto ir_bittest_fehler
    bcf IR_STATUS, 0          ;Если бит равен 1, очистить бит 0
    bsf IR_STATUS, 1          ;в регистре IR_STATUS,
                                ;а бит 1 установить
    bcf IR_STATUS, 2          ;Очистить бит ошибки
    return

ir_bittest_fehler
    bcf IR_STATUS, 0          ;Если возникла ошибка, т. е. было 2 следующих
    bcf IR_STATUS, 1          ;друг за другом одинаковых состояния сигнала,
    bsf IR_STATUS, 2          ;очистить бит 0 и бит 1 в регистре IR_STATUS,
                                ;а бит ошибки (бит 2 в регистре IR_STATUS)
                                ;установить

    return

```

В подпрограмме каждый бит ИК-сигнала проверяется дважды, после чего определяется, какой был перепад в течение одного бита. Если был перепад 0→1, то получен бит, соответствующий логическому 0, а если был перепад 1→0, то получена логическая 1. После того как обе половины бита ИК-сигнала будут исследованы, судя по полученному перепаду, начиная с метки `ir_bittest_low`, определяется, какой бит (0 или 1) был принят. Если же будут получены два следующих подряд одинаковых уровня сигнала, то это соответствует ошибке и в регистре `IR_STATUS` будет установлен бит ошибки (бит 2). Этот регистр можно проверить после возвращения из подпрограммы.

Следующая далее подпрограмма вызывается из программы, чтобы считать бит переключения. После ее вызова проверяется, какое значение имеется у бита переключения. Затем это значение пишется в регистр `IR_TOGGLE`. Если же имеется ошибка, то программа переходит по метке `fehler`.

```

togglebit_lesen
    call IR_Bittest           ;Подпрограмма проверяет следующий бит
    btfsc IR_STATUS, 0
    clrf IR_TOGGLE           ;Бит переключения = 0
    btfsc IR_STATUS, 1
    bsf IR_TOGGLE, 0         ;Бит переключения = 1
    btfsc IR_STATUS, 2
    goto fehler              ;Перейти на подпрограмму обработки ошибки
    decf IR_COUNTER          ;Уменьшить на 1 содержимое счетчика IR_COUNTER
    btfsc STATUS, Z           ;Проверить, сосчитал ли счетчик до 0,
                             ;выполняя счет в обратном порядке
    goto fehler              ;Перейти на подпрограмму обработки ошибки

```

Во время проверки бита переключения после уменьшения содержимого счетчика битов IR_COUNTER на 1 проверяется, досчитал ли он до 0. Если программа работает без ошибки, то счетчик на этом месте еще не должен достигать 0.

```

    movlw d'5'               ;Задать количество разрядов (=5) для
    movwf COUNTER            ;считывания адреса
    clrf IR_ADRESSE
    bcf STATUS, C             ;Очистить бит переноса, чтобы он
                             ;не мешал при сдвиге
adressbits_lesen             ;Чтение битов адреса
    rlf IR_ADRESSE
    call IR_Bittest           ;Подпрограмма проверяет следующий бит
    btfsc IR_STATUS, 0
    bcf IR_ADRESSE, 0         ;Бит адреса = 0
    btfsc IR_STATUS, 1
    bsf IR_ADRESSE, 0         ;Бит адреса = 1
    btfsc IR_STATUS, 2
    goto fehler              ;Приступить к обработке ошибок
    decf IR_COUNTER
    btfsc STATUS, Z           ;Проверить, сосчитал ли счетчик
                             ;IR_COUNTER до 0
    goto fehler              ;Приступить к обработке ошибок
    decfsz COUNTER
    goto adressbits_lesen     ;Если не все биты адреса еще
                             ;считаны, продолжить чтение битов адреса

```

После анализа бита переключения проверяют 5 разрядов адреса. Для этого регистр COUNTER загружается значением 5. Подпрограмма IR_Bittest вызывается до тех пор, пока все разряды адреса не будут считаны. После проверки каждого бита уже считанные биты в регистре IR_ADRESSE сдвигаются влево на разряд. В конце цикла считанное значение адреса будет находиться в регистре IR_ADRESSE.

В это время уже 8 битов будет считано и останется еще 6 битов команды. Проверка и считывание битов команды выполняются аналогично проверке и считыванию адреса.

```

movlw d'6'           ;Задать количество разрядов (=6) для
movwf COUNTER        ;считывания команды
clrf IR_BEFEHL
bcf STATUS, C        ;Очистить бит переноса, чтобы он
                     ;не мешал при сдвиге

befehlbits_lesen
    rlf IR_BEFEHL
    call IR_Bittest   ;Подпрограмма проверяет следующий бит
    btfsc IR_STATUS, 0
    bcf IR_BEFEHL, 0  ;Бит команды = 0
    btfsc IR_STATUS, 1
    bsf IR_BEFEHL, 0  ;Бит команды = 1
    btfsc IR_STATUS, 2
    goto fehler       ;Перейти к обработке ошибок
    decf IR_COUNTER   ;Уменьшить на 1 содержимое счетчика IR_COUNTER
    btfsc STATUS, Z   ;Проверить, досчитал ли
                     ;счетчик до 0
    goto ende_datan   ;Счетчик досчитал до 0
    decfsz COUNTER
    goto befehlbits_lesen ;Если не все биты команды еще
                     ;считаны, продолжить чтение битов команды

```

После того как цикл будет пройден 6 раз, код нажатой кнопки будет храниться в регистре IR_BEFEHL. В это время все 14 битов будут считаны и можно приступить к анализу отдельных регистров.

В приведенном примере оценивается только регистр для хранения команды IR_BEFEHL. Поэтому адрес может выбираться любым. То есть безразлично, нажата ли была кнопка на дистанционном управлении для спутникового приемника или телевизора.

```

ende_datan           ;Все биты данных считались и
                     ;теперь анализируются
movlw VOL_P          ;Значение для кнопки "Громкость +" = 0x10 (VOL +)
subwf IR_BEFEHL, W
btfsc STATUS, Z
goto led1_an         ;Включить LED 1
movlw VOL_M          ;Значение для кнопки "Громкость -" = 0x11 (VOL -)
subwf IR_BEFEHL, W
btfsc STATUS, Z

```

```

goto led2_an      ;Включить LED 2
goto led3_an      ;Включить LED 3, если была нажата
                  ;другая кнопка

```

При анализе содержимого регистра IR_BEFENL оно сравнивается с командами для кнопок "Громкость +" (VOL +) и "Громкость –" (VOL –). Если же определалась кнопка "Громкость +" (VOL +), то происходит переход на метку led1_an и включается светодиод LED 1. При нажатой кнопке "Громкость –" (VOL –) включается светодиод LED 2, а при любой другой кнопке LED 3.

```

led1_an           ;Включение светодиода LED 1 на 1 секунду
bsf LED_1         ;Включить светодиод LED 1
bcf LED_2         ;Выключить светодиод LED 2
bcf LED_3         ;Выключить светодиод LED 3
bcf LED_4         ;Выключить светодиод LED 4
_DELAY_TMR1_US d'500000'
_DELAY_TMR1_US d'500000'
goto main         ;Код опознан, вернуться назад
                  ;в основную программу

Led2_an           ;Включение светодиода LED 2 на 1 секунду
bcf LED_1         ;Выключить светодиод LED 1
bsf LED_2         ;Включить светодиод LED 2
bcf LED_3         ;Выключить светодиод LED 3
bcf LED_4         ;Выключить светодиод LED 4
_DELAY_TMR1_US d'500000'
_DELAY_TMR1_US d'500000'
goto main         ;Код опознан, вернуться назад
                  ;в основную программу

Led3_an           ;Включение светодиода LED 3 на 1 секунду
bcf LED_1         ;Выключить светодиод LED 1
bcf LED_2         ;Выключить светодиод LED 2
bsf LED_3         ;Включить светодиод LED 3
bcf LED_4         ;Выключить светодиод LED 4
_DELAY_TMR1_US d'500000'
_DELAY_TMR1_US d'500000'
goto main         ;Определен неизвестный код, вернуться назад
                  ;в основную программу

```

Согласно приведенным меткам включается только соответствующий светодиод. После следующей далее односекундной паузы снова выполняется возврат в основную программу и опять опрашивается выходной сигнал ИК-приемника.

При возникновении ошибки во время проверки отдельных битов происходит переход на метку fehler.

fehler

```
_DELAY_TMR1_US d'40000' ;Ожидание 40 мс после ошибки, вместе  
                           ;с тем на следующие сигналы больше  
                           ;не будет обращаться внимание  
goto main                 ;Вернуться назад в основную программу
```

В результате программа ожидает 40 мс, пока все данные ИК-телеуправления не будут посланы, и после этого возвращается в основную программу.

Приложение

Распределение в памяти регистров микроконтроллера PIC16F876A

Банк 0		Банк 1		Банк 2		Банк 3	
Адрес	Имя	Адрес	Имя	Адрес	Имя	Адрес	Имя
0x00	Регистр косвенной адресации	0x80	Регистр косвенной адресации	0x100	Регистр косвенной адресации	0x180	Регистр косвенной адресации
0x01	TMR0	0x81	OPTION REG	0x101	TMR0	0x181	OPTION REG
0x02	PCL	0x82	PCL	0x102	PCL	0x182	PCL
0x03	STATUS	0x83	STATUS	0x103	STATUS	0x183	STATUS
0x04	FSR	0x84	FSR	0x104	FSR	0x184	FSR
0x05	PORTA	0x85	TRISA	0x105		0x185	
0x06	PORTB	0x86	TRISB	0x106	PORTB	0x186	TRISB
0x07	PORTC	0x87	TRISC	0x107		0x187	
0x08	PORTD	0x88	TRISD	0x108		0x188	
0x09	PORTE	0x89	TRISE	0x109		0x189	
0x0A	PCLATH	0x8A	PCLATH	0x10A	PCLATH	0x18A	PCLATH
0x0B	INTCON	0x8B	INTCON	0x10B	INTCON	0x18B	INTCON
0x0C	PIR1	0x8C	PIE1	0x10C	EEDATA	0x18C	EECON1
0x0D	PIR2	0x8D	PIE2	0x10D	EEADR	0x18D	EECON2
0x0E	TMR1L	0x8E	PCON	0x10E	EEDATH	0x18E	
0x0F	TMR1H	0x8F		0x10F	EEADRH	0x18F	
0x10	T1CON	0x90		0x110	Регистры общего назначения	0x190	Регистры общего назначения
0x11	TMR2	0x91	SSPCON2	0x111		0x191	
0x12	T2CON	0x92	PR2	0x112		0x192	
0x13	SSPBUF	0x93	SSPADD	0x113		0x193	
0x14	SSPCON	0x94	SSPSTAT	0x114		0x194	
0x15	CCPR1L	0x95		0x115		0x195	
0x16	CCPR1H	0x96		0x116		0x196	
0x17	CCP1CON	0x97		0x117		0x197	
0x18	RCSTA	0x98	TXSTA	0x118		0x198	
0x19	TXREG	0x99	SPBRG	0x119		0x199	
0x1A	RCREG	0x9A		0x11A	Регистры общего назначения	0x19A	Регистры общего назначения
0x1B	CCPR2L	0x9B		0x11B		0x19B	
0x1C	CCPR2H	0x9C	CMCON	0x11C		0x19C	
0x1D	CCP2CON	0x9D	CVRCON	0x11D		0x19D	
0x1E	ADRESH	0x9E	ADRESL	0x11E		0x19E	
0x1F	ADCON0	0x9F	ADCON1	0x11F		0x19F	
0x20	Регистры общего назначения	0xA0	Регистры общего назначения	0x120	Регистры общего назначения	0x1A0	Регистры общего назначения
0x6F	Регистры общего назначения	0xEF		0x16F		0x1EF	
0x70	Регистры общего назначения	0xF0	Доступ к 0x70—0x7F	0x170	Доступ к 0x70—0x7F	0x1F0	Доступ к 0x70—0x7F
0x7F	Регистры общего назначения	0xFF		0x17F		0x1FF	

Обзор регистров управления и состояния

В следующей таблице показаны самые важные регистры специального назначения с перечислением всех наименований битов. Наименования битов соответствуют обозначениям, которые приводятся в Include-файле P16F876A.INC.

Адрес	Имя	Бит 7	Бит 6	Бит 5	Бит 4	Бит 3	Бит 2	Бит 1	Бит 0
Банк 0									
0x03	STATUS	IRP	RP1	RP0	NOT_TO	NOT_PD	Z	DC	C
0x0B	INTCON	GIE	PEIE	TMR0IE	INTE	RBIE	TMR0IF	INTF	RBIF
0x0C	PIR1	PSPIF	ADIF	RCIF	TXIF	SSPIF	CCPIF	TMR2IF	TMR1IF
0x0D	PIR2	—	CMIF	—	EEIF	BCLIF	—	—	CCP2IF
0x10	TICON	—	—	TICKPS1	TICKPS0	TIOSCEN	NOT_TISYNC	TMR1CS	TMR1ON
0x12	T2CON	—	—	TOUTPS3	TOUTPS2	TOUTPS1	TMR2ON	T2CKPS1	T2CKPS0
0x14	SSPCON	WCOL	SSPOV	SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0
0x17	CCP1CON	—	—	CCPIX	CCPIY	CCPIM3	CCPIM2	CCPIM1	CCPIM0
0x18	RCSTA	SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D
0x1D	CCP2CON	—	—	CCP2X	CCP2Y	CCP2M3	CCP2M2	CCP2M1	CCP2M0
0x1F	ADCON0	ADCS1	ADCS0	CHS2	CHS1	CHS0	GO_DONE	—	ADON
Банк 1									
0x81	OPTION_REG	NOT_RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
0x8C	PIE1	PSPIE	ADIE	RCIE	TXIE	SSPIE	CCPIE	TMR2IE	TMR1IE
0x8D	PIE2	—	CMIE	—	EEIE	BCLIE	—	—	CCP2IE
0x8E	PCON	—	—	—	—	—	—	NOT_POR	NOT BOR
0x91	SSPCON2	GCEN	ACKSTAT	ACKDT	ACKEN	RCEN	PEN	RSEN	SEN
0x94	SSPSTAT	SMP	CKE	D_A	P	S	R_W	UA	BF
0x98	TXSTA	CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D
0x9C	CMCON	C2OUT	C1OUT	C2INV	C1INV	CIS	CM2	CM1	CM0
0x9D	CVRCON	CVREN	CVROE	CVRR	—	CVR3	CVR2	CVR1	CVR0
0x9F	ADCON1	ADFM	ADCS2	—	—	PCFG3	PCFG2	PCFG1	PCFG0
Банк 3									
0x18C	EECON1	EEPGRD	—	—	—	WRERR	WREN	WR	RD

Регистр состояния — STATUS

В регистре состояния находятся различные сведения о состоянии микроконтроллера после выполнения математических или логических операций и о том, какой банк регистра активен.

STATUS (адрес 0x03, 0x83, 0x103 или 0x183)

R/W (0)	R/W (0)	R/W (0)	R (1)	R (1)	R/W (x)	R/W (x)	R/W (x)
IRP	RP1	RP0	NOT_TO	NOT_PD	Z	DC	C
7	6	5	4	3	2	1	0

- Бит 7: **IRP**: Бит выбора банка для регистра при косвенной адресации
 1 = банк 2, 3 (0x100—0x1FF)
 0 = банк 0, 1 (0x00—0xFF)
- Бит 6-5: **RP1:RP0**: Бит выбора банка регистра при непосредственной адресации
 11 = банк 3 (0x180—0x1FF)
 10 = банк 2 (0x100—0x17F)
 01 = банк 1 (0x80—0xFF)
 00 = банк 0 (0x00—0x7F)
 Каждый банк содержит 128 байтов
- Бит 4: **NOT_TO**: Флаг переполнения сторожевого таймера
 1 = после включения или выполнения команд CLRWDТ и SLEEP
 0 = после переполнения сторожевого таймера WDT
- Бит 3: **NOT_PD**: Флаг включения питания
 1 = после включения питания или выполнения команды CLRWDТ
 0 = после выполнения команды SLEEP
- Бит 2: **Z**: Флаг нулевого результата
 1 = нулевой результат математической или логической операции
 0 = не нулевой результат математической или логической операции
- Бит 1: **DC**: Флаг десятичного переноса/заема (заем имеет инверсное значение)
 1 = был перенос из младшего полубайта (из бита 3 в бит 4)
 0 = не было переноса из младшего полубайта
- Бит 0: **C**: Флаг переноса/заема (заем имеет инверсное значение)
 1 = был перенос из старшего бита
 0 = не было переноса из старшего бита

Регистр опций — OPTION_REG

В регистре OPTION_REG находятся различные биты управления для конфигурирования и сторожевого таймера. С его помощью можно настроить фронт импульса для прерывания и включить подтягивающие резисторы на входах PORTB.

OPTION_REG (адрес 0x81 или 0x181)

R/W (1)	R/W (1)	R/W (1)	R/W (1)	R/W (1)	R/W (1)	R/W (1)	R/W (1)
NOT_RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
7	6	5	4	3	2	1	0

Бит 7: **NOT_RBPU**: Включение подтягивающих резисторов на входах PORTB

1 = подтягивающие резисторы на входах PORTB отключены

0 = подтягивающие резисторы на входах PORTB включены

Бит 6: **INTEDG**: Бит выбора активного фронта импульса на входе внешнего прерывания INT

1 = прерывание вызывается по переднему фронту сигнала на выводе RB0/INT

0 = прерывание вызывается по заднему фронту сигнала на выводе RB0/INT

Бит 5: **T0CS**: Бит выбора тактового сигнала таймера Timer0 (TMR0)

1 = внешний тактовый сигнал с вывода RA4/T0CKI

0 = внутренний тактовый сигнал

Бит 4: **T0SE**: Бит выбора фронта для внешнего тактового сигнала таймера Timer0 (TMR0)

1 = увеличение по заднему фронту сигнала (с высокого к низкому логическому уровню) на выводе RA4/T0CKI

0 = увеличение по переднему фронту сигнала (с низкого к высокому логическому уровню) на выводе RA4/T0CKI

Бит 3: **PSA**: Выбор включения предварительного делителя частоты

1 = предделитель включен перед сторожевым таймером (WDT)

0 = предделитель выключен перед таймером Timer0 (TMR0)

Бит 2-0: **PS2:PS0**: Выбор коэффициента деления предделителя

Значение бита	TMR0	WDT
000	1 : 2	1 : 1
001	1 : 4	1 : 2
010	1 : 8	1 : 4
011	1 : 16	1 : 8
100	1 : 32	1 : 16
101	1 : 64	1 : 32
110	1 : 128	1 : 64
111	1 : 256	1 : 128

Регистр контроля прерываний — INTCON

В регистре INTCON находятся различные биты разрешения и флаги контроля прерываний.

INTCON (адрес 0x8, 0x8B, 0x10B или 0x18B)

R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (x)
GIE	PEIE	TMR0IE	INTE	RBIE	TMR0IF	INTF	RBIF
7	6	5	4	3	2	1	0

- Бит 7: **GIE**: Бит глобального разрешения прерывания
 1 = разрешены все немаскируемые прерывания
 0 = блокировка всех прерываний
- Бит 6: **PEIE**: Бит разрешения прерываний от периферийных модулей
 1 = разрешены все немаскируемые прерывания от периферии
 0 = блокировка всех прерываний от периферийных модулей
- Бит 5: **TMR0IE**: Бит разрешения прерывания при переполнении таймера Timer0 (TMR0)
 1 = прерывание от TMR0 разрешено
 0 = прерывание от TMR0 запрещено
- Бит 4: **INTE**: Бит разрешения прерывания с вывода RB0/INT
 1 = прерывание разрешено
 0 = прерывание запрещено
- Бит 3: **RBIE**: Бит разрешения прерывания при смене сигнала на входах RB7:RB4 PORTB
 1 = прерывание разрешено
 0 = прерывание запрещено
- Бит 2: **TMR0IF**: Флаг прерывания по переполнению таймера Timer0 (TMR0)
 1 = регистр TMR0 переполнен (сбрасывается программно)
 0 = регистр TMR0 не переполнен
- Бит 1: **INTF**: Флаг внешнего прерывания с вывода RB0/INT
 1 = внешнее прерывание произошло (сбрасывается программно)
 0 = внешнего прерывания с вывода RB0/INT не было
- Бит 0: **RBIF**: Флаг прерывания при смене уровня сигнала на выходах RB7:RB4 PORTB
 1 = зафиксировано изменение уровня сигнала минимум на одном из входов RB7:RB4 (сбрасывается программно)
 0 = изменений уровня сигнала ни на одном из входов RB7:RB4 не было

Первый регистр прерывания от периферии — PIR1

В регистре PIR1 устанавливаются индивидуальные флаги прерываний от периферии.

PIR1 (адрес 0x0C)

R/W (0)	R/W (0)	R (0)	R (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)
PSPIF	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF
7	6	5	4	3	2	1	0

- Бит 7: PSPIF:** Флаг прерывания записи/чтения от ведомого параллельно порта
 1 = произошла операция записи или чтения (сбрасывается программно)
 0 = операция записи или чтения отсутствовала
 В микроконтроллере PIC16F876A этот бит зарезервирован (не используется) и всегда должен содержать 0
- Бит 6: ADIF:** Флаг прерывания от модуля АЦП (A/D-преобразователя)
 1 = преобразование АЦП завершено
 0 = преобразование АЦП не завершило
- Бит 5: RCIF:** Флаг прерывания от приемника USART
 1 = буфер приемника USART полон
 0 = буфер приемника USART пуст
- Бит 4: TXIF:** Флаг прерывания от передатчика USART
 1 = буфер передатчика USART пуст
 0 = буфер передатчика USART полон
- Бит 3: SSPIF:** Флаг прерывания от модуля последовательного синхронного порта MSSP (Master Synchronous Serial Port)
 1 = выполнено условие возникновения прерывания от модуля MSSP (сбрасывается программно, прежде чем происходит выход из программы сервиса прерывания)
 0 = условия возникновения прерывания от модуля MSSP не было
- Бит 2: CCP1IF:** Флаг прерывания от первого модуля захвата/сравнения/ШИМ CCP1 (Capture/Compare/PWM)
Режим захвата:
 1 = прерывание от модуля таймера TMR1 (сбрасывается программно)
 0 = прерывания от модуля таймера TMR1 не было
Режим сравнения:
 1 = значение таймера TMR1 достигло указанного в регистрах CCP1H:CCP1L (сбрасывается программно)
 0 = значение таймера TMR1 не достигло указанного в регистрах CCP1H:CCP1L
Режим ШИМ:
 В режиме ШИМ (PWM) не используется
- Бит 1: TMR2IF:** Флаг прерывания при соответствии значений в регистрах TMR2 и PR2
 1 = содержимое регистров TMR2 и PR2 совпадают (сбрасывается программно)
 0 = содержимое регистров TMR2 и PR2 не совпадают
- Бит 0: TMR1IF:** Флаг прерывания при переполнении регистра TMR1
 1 = переполнение в регистре TMR1 (сбрасывается программно)
 0 = переполнение в регистре TMR1 отсутствует

Второй регистр прерывания от периферии — PIR2

В регистре PIR2 содержатся остальные флаги прерывания от периферии.

PIR2 (адрес 0x0D)

U (0)	R/W (0)	U (0)	R/W (0)	R/W (0)	U (0)	U (0)	R/W (0)
—	CMIF	—	EEIF	BCLIF	—	—	CCP2IF
7	6	5	4	3	2	1	0

Бит 7: Не реализован (англ. Unimplemented), читается как 0

Бит 6: **CMIF**: Флаг прерывания от компаратора

1 = прерывания при изменении на входе компаратора (сбрасывается программно)

0 = вход компаратора не изменился

Бит 5: Не реализован, читается как 0

Бит 4: **EEIF**: Флаг прерывания при операции записи в EEPROM-память

1 = запись в EEPROM-память завершена (сбрасывается программно)

0 = запись еще не завершена или еще не была начата

Бит 3: **BCLIF**: Флаг прерывания при возникновении коллизии на шине I²C

1 = обнаружена коллизия на шине (только в режиме ведущего I²C)

0 = коллизий не обнаружено

Бит 2-1: Не реализованы, читаются как 0

Бит 0: **CCP2IF**: Флаг прерывания от второго модуля захвата/сравнения/ШИМ CCP2 (Capture/Compare/PWM)

Режим захвата:

1 = выполнен захват значения регистра TMR1 (сбрасывается программно)

0 = захвата значения регистра TMR1 не было

Режим сравнения:

1 = значение регистра TMR1 совпало с содержимым регистров CCP2H:CCP2L (сбрасывается программно)

0 = значение регистра TMR1 не совпало с содержимым регистров CCP2H:CCP2L

Режим ШИМ:

В режиме ШИМ (PWM) не используется

Регистр разрешения периферийных прерываний — PIE1

В регистре PIE1 прерывания могут активизироваться индивидуально.

PIE1 (адрес 0x8C)

R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)
PSPIE	ADIE	RCIE	TXIE	SSPIE	CCPIE	TMR2IE	TMR1IE
7	6	5	4	3	2	1	0

- Бит 7: **PSPIE**: Разрешение прерывания записи/чтения ведомого параллельно порта
 1 = прерывание разрешено
 0 = прерывание запрещено
 В микроконтроллере PIC16F876A этот бит не используется и всегда должен содержать 0
- Бит 6: **ADIE**: Разрешение прерывания по окончании преобразования АЦП
 1 = прерывание разрешено
 0 = прерывание запрещено
- Бит 5: **RCIE**: Разрешение прерывания от приемника USART
 1 = прерывание разрешено
 0 = прерывание запрещено
- Бит 4: **TXIE**: Разрешение прерывания от передатчика USART
 1 = прерывание разрешено
 0 = прерывание запрещено
- Бит 3: **SSPIE**: Разрешение прерывания от модуля синхронного последовательного порта SSP (Synchronous Serial Port)
 1 = прерывание разрешено
 0 = прерывание запрещено
- Бит 2: **CCPIE**: Разрешение прерывания от первого модуля захвата/сравнения/ШИМ CCP1 (Capture/Compare/PWM)
 1 = прерывание разрешено
 0 = прерывание запрещено
- Бит 1: **TMR2IE**: Разрешение прерывания при соответствии значений регистров TMR2 и PR2
 1 = прерывание разрешено
 0 = прерывание запрещено
- Бит 0: **TMR1IE**: Разрешение прерывания при переполнении регистра TMR1
 1 = прерывание разрешено
 0 = прерывание запрещено

Регистр разрешения периферийных прерываний — PIE2

В регистре PIE2 содержатся флаги разрешения прерываний от компаратора, EEPROM-памяти и модуля CCP2 (Capture-Compare-PWM).

PIE2 (адрес 0x8D)

U (0)	R/W (0)	U (0)	R/W (0)	R/W (0)	U (0)	U (0)	R/W (0)
—	CMIE	—	EEIE	BCLIE	—	—	CCP2IE
7	6	5	4	3	2	1	0

Бит 7: Не реализован, читается как 0

Бит 6: **CMIE**: Разрешение прерывания от компаратора

1 = прерывание разрешено

0 = прерывание запрещено

Бит 5: Не реализован, читается как 0

Бит 4: **EEIE**: Разрешение прерывания по окончании записи в EEPROM-память

1 = прерывание разрешено

0 = прерывание запрещено

Бит 3: **BCLIE**: Разрешение прерывания при возникновении коллизий на шине I²C

1 = прерывание разрешено

0 = прерывание запрещено

Бит 2-1: Не реализованы, читаются как 0

Бит 0: **CCP2IE**: Разрешение прерывания от модуля CCP2

1 = прерывание разрешено

0 = прерывание запрещено

Регистр контроля питания — PCON

Регистр PCON (Power Control) содержит флаги, с помощью которых можно определить причину сброса микроконтроллера: после включения питания POR (Power-on-Reset), после снижения напряжения питания BOR (Brown-out-Reset), сброса от сторожевого таймера WDT (Watchdog Timer Reset) или внешнего сброса с вывода #MCLR (, и внешний Reset).

PCON (адрес 0x8E)

U (0)	U (0)	U (0)	U (0)	U (0)	U (0)	R/W (0)	R/W (1)
						NOT_POR	NOT_BOR
7	6	5	4	3	2	1	0

Бит 7-2: Не реализованы, читаются как 0

Бит 1: **NOT_POR**: Бит состояния сброса по включению питания POR (Power-on-Reset)

1 = сброса по включению питания не было

0 = был сброс по включению питания (сбрасывается программно)

Бит 0: **NOT_BOR**: Бит состояния сброса по снижению напряжения питания BOR (Brown-Out Reset)

1 = сброса по снижению напряжения питания не было

0 = был сброс по снижению напряжения питания (сбрасывается программно)

Регистр управления модулем таймера 1 — T1CON

В регистре T1CON (Timer 1 Control) устанавливаются коэффициенты деления и различные биты управления.

T1CON (адрес 0x10)

U (0)	U (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)
—	—	TICKPS1	TICKPS0	TIOSCEN	NOT_T1SYNC	TMR1CS	TMR1ON
7	6	5	4	3	2	1	0

Бит 7-6: Не реализованы, читаются как 0

Бит 5-4: **TICKPS1:TICKPS0**: Биты выбора для делителя таймера 1

11 = коэффициент деления 1 : 8

10 = коэффициент деления 1 : 4

01 = коэффициент деления 1 : 2

00 = коэффициент деления 1 : 1

Бит 3: **TIOSCEN**: Включение тактового генератора таймера 1

1 = генератор включен

0 = генератор выключен

Бит 2: **NOT_T1SYNC**: Бит управления синхронизацией внешнего тактового сигнала для таймера 1

Если TMR1CS = 1:

1 = вход для внешнего тактирующего сигнала не синхронизировать

0 = вход для внешнего тактирующего сигнала синхронизировать

Если TMR1CS = 0:

Бит игнорируется. Таймер 1 использует внутренний тактовый сигнал

Бит 1: **TMR1CS**: Бит выбора источника тактового сигнала для таймера 1

1 = внешний источник с вывода RC0/TIOSO/TICK1 (передний фронт)

0 = внутренний тактовый сигнал (Fosc/4)

Бит 0: **TMR1ON**: Включение и выключение модуля таймера 1

1 = таймер включен

0 = таймер выключен

Регистр управления модулем таймера 2 — T2CON

С помощью регистра T2CON осуществляется управление таймером 2.

T2CON (адрес 0x12)

U (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)
—	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0
7	6	5	4	3	2	1	0

Бит 7: Не реализован, читается как 0

Бит 6-3: **TOUTPS3:TOUTPS0**: Выбор коэффициента деления для выходного делителя таймера 2 (Timer 2)

0000 = коэффициент деления 1 : 1

0001 = коэффициент деления 1 : 2

0010 = коэффициент деления 1 : 3

...

1111 = коэффициент деления 1 : 16

Бит 2: **TMR2ON**: Включение и выключение модуля таймера 2

1 = таймер 2 включен

0 = таймер 2 выключен

Бит 1;0: **T2CKPS1:T2CKPS0**: выбор коэффициента деления для предделителя таймера 2

00 = коэффициент деления 1:1

01 = коэффициент деления 1:4

10 = коэффициент деления 1:16

11 = коэффициент деления 1:16

Регистр состояния модуля MSSP — SSPSTAT (режим SPI)

Биты в регистре SSPSTAT дают представление о состоянии модуля синхронного последовательного порта MSSP. Значение битов зависит от выбранного режима работы модуля: последовательного периферийного интерфейса SPI (Serial Peripheral Interface) или шины I²C (Inter-Integrated Circuit).

SSPSTAT (адрес 0x94)

R/W (0)	R/W (0)	R (0)	R (0)	R (0)	R (0)	R (0)	R (0)
SMP	CKE	D_A	P	S	R_W	UA	BF
7	6	5	4	3	2	1	0

Бит 7: **SMP**: Бит момента выборки

Ведущий режим SPI:

1 = опрос входа в конце периода вывода данных

0 = опрос входа в середине периода вывода данных

Ведомый режим SPI:

Для режима ведомого SPI бит SMP всегда должен быть сброшен в 0

Бит 6: **CKE**: Бит выбора фронта тактового сигнала для режима SPI

1 = данные передаются по переднему фронту сигнала на выводе SCK

0 = данные передаются по заднему фронту сигнала на выводе SCK

Полярность тактового сигнала устанавливается с помощью бита CKP регистра SSPCON

Бит 5: **D_A**: Бит данные/адрес

Используется только в режиме I²C

Бит 4: **P**: Бит STOP

Используется только в режиме I²C

Бит 3: **S**: Бит START

Используется только в режиме I²C

Бит 2: **R_W**: Бит чтения/записи

Используется только в режиме I²C

Бит 1: **UA**: Флаг обновления адреса (Update Address) устройства

Используется только в режиме I²C

Бит 0: **BF**: Бит состояния буфера (только в режиме приемника)

1 = прием завершен (буфер SSPBUF полон)

0 = прием не завершен (буфер SSPBUF пуст)

Регистр состояния модуля MSSP — SSPSTAT (в режиме I²C)

Биты регистра SSPSTAT дают представление о состоянии модуля MSSP. Значение битов зависит от выбранного режима работы модуля: последовательного периферийного интерфейса SPI (Serial Peripheral Interface) или шины I²C (Inter-Integrated Circuit).

SSPSTAT (адрес 0x94)

R/W (0)	R/W (0)	R (0)	R (0)	R (0)	R (0)	R (0)	R (0)
SMP	CKE	D_A	P	S	R_W	UA	BF
7	6	5	4	3	2	1	0

- Бит 7: **SMP:** Бит управления скоростью нарастания уровня сигнала
Ведущий или ведомый режим:
 1 = управление скоростью нарастания сигнала включено при стандартной скорости обмена (100 кГц и 1 МГц)
 0 = управление скоростью нарастания сигнала включено в скоростном режиме (400 кГц)
- Бит 6: **CKE:** Бит выбора спецификации протокола SMBus (System Management Bus)
Ведущий или ведомый режим:
 1 = входные уровни соответствуют спецификации протокола SMBus
 0 = запрет спецификации протокола SMBus для входных уровней
- Бит 5: **D_A:** Бит данных/адрес (только для режима I²C)
Ведущий режим:
 Зарезервировано
Ведомый режим:
 1 = показывает, что последний принятый или переданный байт относится к данным
 0 = показывает, что последний принятый или переданный байт относится к адресу
- Бит 4: **P:** Бит STOP (только для режима I²C)
 1 = указывает, что бит STOP был обнаружен последним
 0 = бит STOP не является последним
- Бит 3: **S:** Бит START (только для режима I²C)
 1 = указывает, что бит START был обнаружен последним
 0 = бит START не является последним
 Этот бит очищается при сбросе и при сбросе бита SSPEN в регистре SSPCON
- Бит 2: **R_W:** Бит чтения/записи (только для режима I²C)
 Значение бита действительно только после совпадения адреса и до приема бита START, STOP или #ACK.
Ведомый режим:
 1 = чтение
 0 = запись
Ведущий режим:
 1 = осуществляется передача данных
 0 = передачи данных нет

- Бит 1: **UA**: Флаг обновления адреса устройства (только для 10-битного ведомого режима)
1 = показывает, что пользователь должен обновить адрес в регистре SSPADD
0 = обновление адреса не требуется
- Бит 0: **BF**: Бит состояния заполнения буфера
- В режиме приема:
1 = прием завершен, буфер SSPBUF полон
0 = прием не завершен, буфер SSPBUF пуст
- В режиме передачи:
1 = выполняется передача данных (исключая биты #ACK и STOP), буфер SSPBUF полон
0 = передача данных завершена (исключая биты #ACK и STOP), буфер SSPBUF пуст

Регистр управления модулем MSSP — SSPCON (режим SPI)

С помощью регистра SSPCON осуществляются специальные настройки для выполнения передачи. Значения битов зависят от выбранного режима работы модуля MSSP (SPI или I²C).

SSPCON (адрес 0x14)

R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)
WCOL	SSPOV	SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0
7	6	5	4	3	2	1	0

Бит 7: **WCOL**: Бит распознавания конфликта данных при записи (только в режиме передачи)

1 = запись в регистр SSPBUF во время передачи предыдущего байта

0 = конфликта не было

Бит 6: **SSPOV**: Бит переполнения приемника

1 = новый байт принимался, в то время как предыдущий байт находился еще в регистре буфера приемника SSPBUF. Это может происходить только в ведомом режиме. В ведущем режиме этот бит в 1 не устанавливается, поскольку каждая операция инициализируется записью в регистр SSPBUF (сбрасывается программно)

0 = нет переполнения

Бит 5: **SSPEN**: Бит включения модуля MSSP

1 = модуль MSSP включен, выходы SCK, SDO, SDI и #SS используются модулем MSSP

0 = модуль MSSP выключен, выходы SCK, SDO, SDI и #SS используются в качестве цифровых выводов порта ввода/вывода

Бит 4: **CKP**: Бит выбора полярности тактового сигнала

1 = пассивный уровень тактового сигнала — высокий логический уровень

0 = пассивный уровень тактового сигнала — низкий логический уровень

Бит 3-0: **SSPM3:SSPM0**: Биты выбора режима работы модуля MSSP

0101 = ведомый режим SPI, вывод SCK применяется для тактового сигнала, вывод #SS не используется модулем и может применяться в качестве цифрового вывода порта ввода/вывода

0100 = ведомый режим SPI, вывод SCK применяется для тактового сигнала, вывод #SS используется модулем MSSP

0011 = ведущий режим SPI, тактовый сигнал = выход таймера 2/2

0010 = ведущий режим SPI, тактовый сигнал = $F_{osc}/64$

0001 = ведущий режим SPI, тактовый сигнал = $F_{osc}/16$

0000 = ведущий режим SPI, тактовый сигнал = $F_{osc}/4$

Регистр управления модуля MSSP — SSPCON (режим I²C)

С помощью регистра SSPCON осуществляются специальные настройки для выполнения передачи. Значения битов зависят от выбранного режима работы модуля MSSP (SPI или I²C).

SSPCON (адрес 0x14)

R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)
WCOL	SSPOV	SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0
7	6	5	4	3	2	1	0

Бит 7: **WCOL:** Бит распознавания конфликта данных при записи

Ведущий режим (передача):

1 = неудачная попытка записи в регистр SSPBUF в режиме I²C во время отсутствия условий для начала передачи (сбрасывается в 0 программно)

0 = конфликта не было

Ведомый режим (передача):

1 = запись в регистр SSPBUF, во время передачи предыдущего слова

0 = конфликта не было

Ведущий или ведомый режим (прием):

Этот бит при приеме не имеет значения

Бит 6: **SSPOV:** Бит переполнения во время приема

Режим приема:

1 = принят новый байт, в то время как регистр SSPBUF содержит предыдущее значение

0 = нет переполнения

Режим передачи:

Этот бит при передаче не имеет значения

Бит 5: **SSPEN:** Бит включения модуля MSSP

1 = модуль MSSP включен, выводы SDA и SCL используются модулем

0 = модуль MSSP выключен, выводы SDA и SCL используются в качестве цифровых выводов порта ввода/вывода

Бит 4: **CKP:** Бит управления тактовым сигналом на выводе SCL

Ведомый режим:

1 = не управлять тактовым сигналом на выводе SCL

0 = удержание тактового сигнала на выводе SCL на низком логическом уровне (используется для подготовки данных)

Ведущий режим:

Бит в этом режиме не имеет значения

Бит 3-0: **SSPM3:SSPM0:** Биты выбора режима работы модуля MSSP

В 1111 = ведомый режим I²C, 10-битная адресация и разрешение прерывания при приеме стартовых (START) и стоповых (STOP) битов

В 1110 = ведомый режим I²C, 7-битная адресация и разрешение прерывания при приеме стартовых (START) и стоповых (STOP) битов

В 1011 = ведущий режим I²C (ведомый режим выключен)

1000 = ведущий режим I²C, тактовый сигнал = $F_{osc}/(4 * (SSPAD + 1))$

0111 = ведомый режим I²C, 10-битная адресация

0110 = ведомый режим I²C, 7-битная адресация

Второй регистр управления модулем MSSP — SSPCON2 (режим I²C)

В регистре SSPCON2 находятся биты для генерации последовательности запуска и остановки.

SSPCON2 (адрес 0x91)

R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)
GCEN	ACKSTAT	ACKDT	ACKEN	RCEN	PEN	RSEN	SEN
7	6	5	4	3	2	1	0

- Бит 7: **GCEN**: Бит разрешения общего вызова (только для ведомого режима I²C)
 1 = разрешить прерывание при приеме в регистр SSPSR адреса общего вызова 0x0000
 0 = поддержка общего вызова выключена
- Бит 6: **ACKSTAT**: Бит состояния подтверждения (только при передаче в ведущем режиме)
 1 = отсутствие приема подтверждения от ведомого
 0 = подтверждение от ведомого было получено
- Бит 5: **ACKDT**: Бит подтверждения данных (только при передаче ведущего)
 1 = нет послышки подтверждения
 0 = послышка подтверждения
- Бит 4: **ACKEN**: Активизация последовательности подтверждения (только при передаче в ведущем режиме)
 1 = выдача последовательности подтверждения на выводах SDA и SCL и передача бита ACKDT. Бит автоматически сбрасывается микроконтроллером
 0 = подтверждение не формируется
- Бит 3: **RCEN**: Бит разрешения приема (только в режиме ведущего)
 1 = разрешить прием данных в режиме I²C
 0 = прием запретить
- Бит 2: **PEN**: Сформировать на шине I²C стоп-бит (STOP) (только в режиме ведущего)
 1 = на выводах SDA и SCL сформировать сигналы, соответствующие выдаче стоп-бита (STOP). Бит сбрасывается автоматически
 0 = стоп-бит (STOP) не формировать
- Бит 1: **RSEN**: Сформировать на шине I²C бит повторного старта (START) (только в режиме ведущего)
 1 = на выводах SDA и SCL сформировать сигналы, соответствующие выдаче повторного старта (START). Бит сбрасывается в 0 автоматически
 0 = бит повторного старта (START) не формировать
- Бит 0: **SEN**: Сформировать на шине I²C бит старта/расширения тактового сигнала
- Ведущий режим:
 1 = на выводах SDA и SCL сформировать сигналы, соответствующие выдаче старта (START). Бит сбрасывается в 0 автоматически
 0 = стартовый бит (START) не формируется
- Ведомый режим:
 1 = расширение тактового сигнала разрешено для ведомого при приеме и передаче
 0 = расширение тактового сигнала разрешено только при передаче ведомого

Регистр управления модулем Сравнения/Захвата/ШИМ — CCPxCON

С помощью регистров CCP1CON и CCP2CON осуществляется настройка модуля Сравнения/Захвата/ШИМ или иначе CCP (Capture/Compare/PWM).

CCP1CON (адрес 0x17)

CCP2CON (адрес 0x1D)

U (0)	U (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)
—	—	CCP1X	CCP1Y	CCP1M3	CCP1M2	CCP1M1	CCP1M0
—	—	CCP2X	CCP2Y	CCP2M3	CCP2M2	CCP2M1	CCP2M0
7	6	5	4	3	2	1	0

Бит 7-6: Не реализованы, читаются как 0

Бит 5-4: **CCPxX:CCPxY**: Младшие биты скважности ШИМ (PWM)

Режим захвата:

Не используются

Режим сравнения:

Не используются

Режим ШИМ:

Два младших бита скважности ШИМ (PWM). 8 старших битов находятся в регистре CCPxL

Бит 3-0: **CCPxM3:CCPxM0**: Биты выбора режима для соответствующего модуля Сравнения/Захвата/ШИМ (CCPx)

0000 = модуль Сравнения/Захвата/ШИМ (CCPx) выключен

0100 = режим захвата по каждому заднему фронту сигнала

0101 = режим захвата по каждому переднему фронту сигнала

0110 = режим захвата по каждому 4-му переднему фронту сигнала

0111 = режим захвата по каждому 16-му переднему фронту сигнала

1000 = режим сравнения, при соответствии устанавливает выходной сигнал (устанавливается флаг CCPxIF в 1)

1001 = режим сравнения, при соответствии сбрасывает выходной сигнал (устанавливается флаг CCPxIF в 1)

1010 = режим сравнения, при соответствии осуществляется прерывание (устанавливается флаг CCPxIF в 1, на выводы CCPx не влияет)

1011 = режим сравнения, вызывает специальное событие (устанавливается флаг CCPxIF в 1; на вывод CCPx не влияет); CCP1 сбрасывает таймер 1; CCP2 сбрасывает таймер 1 и запускает АЦП-преобразование, если модуль АЦП включен

11xx = режим ШИМ (PWM)

Регистр состояния и управления приемника модуля USART — RCSTA

С помощью регистра RCSTA настраивается приемник модуля универсального синхронно-асинхронного приемопередатчика или иначе USART (Universal Synchronous-Asynchronous Receiver/Transmitter).

RCSTA (адрес 0x18)

R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R (0)	R (0)	R (x)
SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D
7	6	5	4	3	2	1	0

Бит 7: **SPEN**: Разрешение работы последовательного порта

1 = модуль USART включен. Выводы RC7/RX/DT и RC6/TX/CK подключены к модулю USART

0 = модуль USART выключен

Бит 6: **RX9**: Бит разрешения 9-битного приема

1 = выбран 9-битный прием

0 = выбран 8-битный прием

Бит 5: **SREN**: Бит разрешения одиночного приема

Асинхронный режим:

Не имеет значения

Синхронный режим (ведущее устройство):

1 = одиночный прием разрешен

0 = одиночный прием запрещен

Бит сбрасывается в 0 после завершения приема

Синхронный режим — ведомое устройство:

Не имеет значения

Бит 4: **CREN**: Бит разрешения непрерывного приема

Асинхронный режим:

1 = непрерывный прием разрешен

0 = непрерывный прием запрещен

Синхронный режим:

1 = непрерывный прием разрешен до тех пор, пока бит CREN не сброшен (при установке бита CREN автоматически сбрасывается бит SREN)

0 = непрерывный прием запрещен

Бит 3: **ADDEN**: Бит разрешения распознавания адреса

Асинхронный 9-битный прием (RX9 = 1):

1 = распознавание адреса разрешено. Если бит RSR<8> = 1, то генерируется прерывание и загружается буфер приемника

0 = распознавание адреса запрещено. Принимаются все байты, девятый бит может использоваться в качестве бита для проверки четности

Бит 2: **FERR**: Ошибка кадра (Framing Error)

1 = ошибка кадра (может сбрасываться во время чтения регистра RCREG и получения следующего допустимого байта)

0 = ошибка кадра отсутствует

Бит 1: **OERR**: Ошибка при переполнении внутреннего буфера

1 = произошла ошибка переполнения (бит может быть установлен в 0 во время сброса бита CREN)

0 = ошибки переполнения не было

Бит 0: **RX9D**: 9-й бит принятых данных (может быть также битом четности, должен рассчитываться пользователем)

Регистр состояния и управления передатчика модуля USART — TXSTA

С помощью регистра TXSTA настраивается передатчик универсального синхронно-асинхронного приемопередатчика или иначе USART (Universal Synchronous-Asynchronous Receiver/Transmitter).

TXSTA (адрес 0x98)

R/W (0)	R/W (0)	R/W (0)	R/W (0)	U (0)	R/W (0)	R (1)	R/W (0)
CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D
7	6	5	4	3	2	1	0

Бит 7: **CSRC**: Выбор источника тактового сигнала

Асинхронный режим:

Не имеет значения

Синхронный режим:

1 = ведущий, тактовый сигнал формируется от внутреннего генератора скорости обмена в бодах или иначе BRG (Baud Rate Generator)

0 = ведомый (тактовый сигнал поступает от внешнего источника с входа CK)

Бит 6: **TX9**: Бит разрешения 9-битной передачи

1 = 9-битная передача

0 = 8-битная передача

Бит 5: **TXEN**: Бит разрешения передачи

1 = передача разрешена

0 = передача запрещена

Бит 4: **SYNC**: Выбор режима работы модуля USART

1 = синхронный режим

0 = асинхронный режим

Бит 3: Не реализован, читается как 0

Бит 2: **BRGH**: Бит выбора высокоскоростного режима

Асинхронный режим:

1 = высокая скорость

0 = низкая скорость

Синхронный режим:

В этом режиме не имеет значения

Бит 1: **TRMT**: Флаг очистки регистра сдвига передатчика модуля USART

1 = регистр сдвига пуст

0 = регистр сдвига полон

Бит 0: **TX9D**: 9-й бит передаваемых данных, также может использоваться для проверки четности

Регистр управления модулем АЦП — ADCON0

С помощью регистра ADCON0 включается модуль АЦП и выбирается канал для осуществления преобразования.

ADCON0 (адрес 0x1F)

R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	U (0)	R/W (0)
ADCS1	ADCS0	CHS2	CHS1	CHS0	GO_DONE	—	ADON
7	6	5	4	3	2	1	0

Бит 7-6: ADCS1:ADCS0: Выбор источника тактового сигнала для АЦП

ADCON1 <ADCS2>	ADCON0 <ADCS1:ADCS0>	Тактовый сигнал для преобразования
0	00	$F_{osc}/2$
0	01	$F_{osc}/8$
0	10	$F_{osc}/32$
0	11	F_{RC} (внутренний RC-генератор)
1	00	$F_{osc}/4$
1	01	$F_{osc}/16$
1	10	$F_{osc}/64$
1	11	F_{RC} (внутренний RC-генератор)

Бит 5-3: CHS2:CHS0: Выбор аналогового канала

000 = канал 0 (AN0)

001 = канал 1 (AN1)

010 = канал 2 (AN2)

011 = канал 3 (AN3)

100 = канал 4 (AN4)

101 = канал 5 (AN5)

110 = канал 6 (AN6)

111 = канал 7 (AN7)

В PIC16F876A реализованы только каналы от 0 до 4. Поэтому другие каналы для этого типа микроконтроллера выбирать не следует

Бит 2: GO_DONE: Бит состояния модуля АЦП

Если модуль АЦП включен (бит ADON = 1):

1 = модуль АЦП выполняет преобразование аналогового значения. При установке этого бита запускается процесс преобразования, а после его окончания бит автоматически сбрасывается

0 = нет преобразования (состояние ожидания)

Бит 1: Не реализован, читается как 0

Бит 0: ADON: Включение/выключение модуля АЦП

1 = модуль АЦП включен

0 = модуль АЦП выключен и ток не потребляет

Регистр управления модулем АЦП — ADCON1

В регистре ADCON1 выбирают формат результата аналого-цифрового преобразования и опорное напряжение.

ADCON1 (адрес 0x9F)

R/W (0)	R/W (0)	U (0)	U (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)
ADFM	ADCS2	—	—	PCFG3	PCFG2	PCFG1	PCFG0
7	6	5	4	3	2	1	0

Бит 7: ADFM: Выбор формата отображения результата

1 = правое выравнивание, 6 старших битов в регистре ADRESH читаются как 0

0 = левое выравнивание, 6 младших битов в регистре ADRESH читаются как 0

Бит 6: ADCS2: Выбор источника тактового сигнала для АЦП

ADCON1 <ADCS2>	ADCON0 <ADCS1:ADCS0>	Тактовый сигнал для преобразования
0	00	$F_{osc}/2$
0	01	$F_{osc}/8$
0	10	$F_{osc}/32$
0	11	F_{RC} (внутренний RC-генератор)
1	00	$F_{osc}/4$
1	01	$F_{osc}/16$
1	10	$F_{osc}/64$
1	11	F_{RC} (внутренний RC-генератор)

Бит 5-4: Не реализованы, читаются как 0

Бит 3-0: PCFG3:PCFG0: Управляющие биты для настройки каналов АЦП

PCFG [3...0]	AN7	AN6	AN5	AN4	AN3	AN2	AN1	AN0	V _{REF+}	V _{REF-}
0000	A	A	A	A	A	A	A	A	V _{DD}	V _{SS}
0001	A	A	A	A	V _{REF+}	A	A	A	AN3	V _{SS}
0010	D	D	D	A	A	A	A	A	V _{DD}	V _{SS}
0011	D	D	D	A	V _{REF+}	A	A	A	AN3	V _{SS}
0100	D	D	D	D	A	D	A	A	V _{DD}	V _{SS}
0101	D	D	D	D	V _{REF+}	D	A	A	AN3	V _{SS}
0110	D	D	D	D	D	D	D	D	—	—
0111	D	D	D	D	D	D	D	D	—	—
1000	A	A	A	A	V _{REF+}	V _{REF-}	A	A	AN3	AN2
1001	D	D	A	A	A	A	A	A	V _{DD}	V _{SS}
1010	D	D	A	A	V _{REF+}	A	A	A	AN3	V _{SS}
1011	D	D	A	A	V _{REF+}	V _{REF-}	A	A	AN3	AN2
1100	D	D	D	A	V _{REF+}	V _{REF-}	A	A	AN3	AN2
1101	D	D	D	D	V _{REF+}	V _{REF-}	A	A	AN3	AN2
1110	D	D	D	D	D	D	D	A	V _{DD}	V _{SS}
1111	D	D	D	D	V _{REF+}	V _{REF-}	D	A	AN3	AN2

Регистр управления модулем компаратора — CMCON

Регистр CMCON управляет входом компаратора и выходом мультимплексора.

CMCON (адрес 0x9C)

R (0)	R (0)	R/W (0)	R/W (0)	R/W (0)	R/W (1)	R/W (1)	R/W (1)
C2OUT	C1OUT	C2INV	C1INV	CIS	CM2	CM1	CM0
7	6	5	4	3	2	1	0

Бит 7: **C2OUT**: Выход компаратора 2

Если бит C2INV = 0:

1 = $C2 V_{IN+} > C2 V_{IN-}$

0 = $C2 V_{IN+} < C2 V_{IN-}$

Если C2INV = 1:

1 = $C2 V_{IN+} < C2 V_{IN-}$

0 = $C2 V_{IN+} > C2 V_{IN-}$

Бит 6: **C1OUT**: Выход компаратора 1

Если C1INV = 0:

1 = $C1 V_{IN+} > C1 V_{IN-}$

0 = $C1 V_{IN+} < C1 V_{IN-}$

Если C1INV = 1:

1 = $C1 V_{IN+} < C1 V_{IN-}$

0 = $C1 V_{IN+} > C1 V_{IN-}$

Бит 5: **C2INV**: Инвертирование выхода компаратора 2

1 = выход C2 инвертирован

0 = выход C2 не инвертирован

Бит 4: **C1INV**: Инвертирование выхода компаратора 1

1 = выход C1 инвертирован

0 = выход C1 не инвертирован

Бит 3: **CIS**: Бит переключения входов компараторов

Если биты CM2:CM0 = 110:

1 = $C1 V_{IN-}$ соединен с выводом RA3/AN3

$C2 V_{IN-}$ соединен с выводом RA2/AN2

0 = $C1 V_{IN-}$ соединен с выводом RA0/AN0

$C2 V_{IN-}$ соединен с выводом RA1/AN1

Бит 2-0: **CM2:CM0**: Выбор режима компаратора

Отдельные рабочие режимы приведены в технической документации на странице 136 на рисунке 12.1 (FIGURE 12-1). Документация находится на прилагаемом компакт-диске в файле PIC16F876A_DataSheet.pdf.

Регистр управления опорным напряжением компаратора — CVRCON

С помощью регистра CVRCON настраивают опорное напряжение для компаратора.

CVRCON (адрес 0x9D)

R/W (0)	R/W (0)	R/W (0)	U (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)
CVREN	CVROE	CVRR	—	CVR3	CVR2	CVR1	CVR0
7	6	5	4	3	2	1	0

Бит 7: **CVREN**: Бит включения опорного напряжения для компаратора

1 = CV_{REF} включено

0 = CV_{REF} отключено

Бит 6: **CVROE**: Бит включения выхода компаратора CV_{REF}

1 = уровень напряжения выхода компаратора CV_{REF} подключен к выводу RA2/AN2/ V_{REF-}/CV_{REF}

0 = уровень напряжения выхода компаратора CV_{REF} не подключен к выводу RA2/AN2/ V_{REF-}/CV_{REF}

Бит 5: **CVRR**: Бит выбора диапазона напряжения V_{REF} компаратора

1 = от 0 до $0,75 * CV_{RSRC}$ с шириной шага $CV_{RSRC}/24$

0 = от $0,25 * CV_{RSRC}$ до $0,75 * CV_{RSRC}$ с шириной шага $CV_{RSRC}/32$

Бит 4: Не реализован, читается как 0

Бит 3-0: **CVR3:CVR0**: Биты выбора величины напряжения V_{REF} компаратора

Если CVRR = 1:

$CV_{REF} = (VR<3:0> / 24) * (CV_{RSRC})$

Если CVRR = 0:

$CV_{REF} = 1/4 * (CV_{RSRC}) + (VR3:VR0/32) * (CV_{RSRC})$

Регистр управления косвенной записи/чтения EEPROM-памяти данных и Flash-памяти программ — EECON1

Регистр EECON1 содержит биты управления записью или чтением внутренней EEPROM-памяти данных и Flash-памяти программ.

EECON1 (адрес 0x18C)

R/W (x)	U (0)	U (0)	U (0)	R/W (x)	R/W (0)	R/W (0)	R/W (0)
EEPGD	—	—	—	WRERR	WREN	WR	RD
7	6	5	4	3	2	1	0

Бит 7: **EEPGD**: Бит выбора Flash-памяти программ или EEPROM-памяти данных

1 = память программ

0 = память данных

Бит 6-4: Не реализованы, читаются как 0

Бит 3: **WRERR**: Бит флага ошибки записи данных

1 = преждевременное завершение операции записи

0 = операция записи завершена

Бит 2: **WREN**: Бит разрешения записи данных

1 = операция записи разрешена

0 = операция записи запрещена

Бит 1: **WR**: Бит управления записью

1 = начать цикл записи. Бит автоматически сбрасывается, если операция записи завершена. Бит может устанавливаться только программно

0 = цикл записи завершен

Бит 0: **RD**: Бит управления чтением

1 = начать цикл чтения. Бит автоматически сбрасывается, если операция чтения завершена. Бит может устанавливаться только программно

0 = цикл чтения не начат

Список источников информации

1. Michael Hofmann (Михаэль Хофманн)
<http://www.edmh.de> (сайт автора)
2. Microchip
<http://www.microchip.com>
Спецификации, руководства и справочники
3. NXP
<http://www.nxp.com>
Спецификации кодов интерфейсов PC и RC5
4. Electronic Assembly
<http://www.lcd-module.de>
Спецификация на дисплей M EA-DOG
5. Sitronix
<http://www.sitronix.com.tw>
Спецификация на ЖКИ-контроллер
6. Maxim
<http://www.maxim-ic.com>
Спецификация RS-232
7. Vishay
<http://www.vishay.com>
Спецификация на ИК-приемник
8. Cadsoft
<http://www.cadsoft.de>
Программное обеспечение для электрических схем и размещения схемных элементов
9. DL4YHF
<http://www.qsl.net/dl4yhf/>
<http://freenet-homepage.de/dl4yhf/>
Среда программирования WinPic
10. Reichelt Elektronik
Elektronikring 1
26452 Sande
Tel.: 04422 955-333
<http://www.reichelt.de>
Поставщик комплектующих изделий
11. Conrad Electronic
Klaus-Conrad-Str. 1
92240 Hirschau
Tel.: 0180 5312111
<http://www.conrad.de>
Поставщик комплектующих изделий

Описание компакт-диска

Далее в виде таблицы приведено описание компакт-диска.

Папка	Файл	Описание
	\CD_Menue.exe	Меню для работы с компакт-диском
	\Copyright.pdf	Авторские права издательства FRANZIS
\Adobe		Дистрибутивы Adobe AcrobatReader для чтения PDF-файлов
	\AdbeRdr812_de_DE.exe	Дистрибутив для установки Adobe AcrobatReader 8.12 (нем.)
	\AdbeRdr930_ru_RU.exe	Дистрибутив для установки Adobe AcrobatReader 9.3 (рус.)
\Arbeitsunterlagen		Документация для работы с микроконтроллером
	\Befehlsübersicht.pdf	Обзор команд микроконтроллера PIC16F876A (нем.)
	\Befehlsübersicht_ru.pdf	Обзор команд микроконтроллера PIC16F876A (рус.)
	\Registerbeschreibung.pdf	Описание регистров (нем.)
	\Registerbeschreibung_ru.pdf	Описание регистров (рус.)
\Beispiele		Примеры программ из глав книги
	\Anleitung.txt	Инструкция по использованию примеров (нем.)
	\Kurzanleitung_HyperTerminal.pdf	Краткое руководство по работе с ПО HyperTerminal (нем.)
	\Kurzanleitung_HyperTerminal_ru.pdf	Краткое руководство по работе с ПО HyperTerminal (рус.)
	\P16f876a.inc	Стандартный заголовочный файл Microchip Technology, Inc. для микроконтроллера P16F876A (англ.)
	\Vorlage.asm	Шаблон для написания программ на ассемблере (см. разд. 3.4)
	\Инструкция.txt	Инструкция по использованию примеров (нем.+рус.)
	\Инструкция_ru.txt	Инструкция по использованию примеров (нем.+рус.)
\Beispiele\Kap_1_2_6_Unterprogramm		Полные тексты программы из разд. 1.2.6 "Вызов подпрограмм"
\Beispiele\Kap_1_2_7_indAdressierung		Полные тексты программы из разд. 1.2.7 "Косвенная адресация"
\Beispiele\Kap_1_2_8_EEPROM		Полные тексты программы из разд. 1.2.8 "Чтение и запись внутренней EEPROM-памяти"

(продолжение)

Папка	Файл	Описание
\Beispiele\Kap_6_4_LED-Muster		Полные тексты программы из разд. 6.4 "Пример программы "Управление светодиодами""
\Beispiele\Kap_7_2_Lauflicht		Полные тексты программы "Бегущие огни" из разд. 7.2
\Beispiele\Kap_8_2_Voltmeter		Полные тексты программы "Вольтметр" из разд. 8.2
\Beispiele\Kap_9_4_HelloWorld		Полные тексты программы для вывода текста "Hello World!" на ЖК-индикатор, рассмотренной в разд. 9.4
\Beispiele\Kap_10_4_Spannungsanzeige		Полные тексты программы для отображения напряжения на индикаторе, рассмотренной в гл. 10
\Beispiele\Kap_11_P-R-Messung		Полные тексты программы для измерения мощности и сопротивления, рассмотренной в гл. 11
\Beispiele\Kap_12_4_PC-Steuerung		Полные тексты программы для управления микроконтроллером с помощью компьютера посредством последовательного интерфейса, рассмотренной в разд. 12.4
\Beispiele\Kap_13_3_Messwertspeicherung		Полные тексты программы из разд. 13.3 "Пример программы "Сохранение измеренных значений в EEPROM-памяти""
\Beispiele\Kap_14_2_IR-Schalter		Полные тексты программы для инфракрасного управления, рассмотренной в разд. 14.2
\Datenblaetter	\I2C_Spec_NXP_UM10204_3.pdf	Спецификация и руководства пользователя интерфейса I ² C (англ.)
	\PIC16F876A_DataSheet.pdf	Технические характеристики на микроконтроллер PIC16F876A (англ.)
	\PIC16F87XA_Errata.pdf	Замеченные опечатки в документации по семейству микроконтроллеров PIC16F87XA (англ.)
	\PICmicro_MidRange_ReferenceManual_33023a.pdf	Расширенное руководство по среднему семейству микроконтроллеров PICmicro (англ.)
\Entwicklungsboard		Документация и ПО для изготовления монтажной платы
	\Bestueckungsplan_PIC-Eval.pdf	Схема расположения деталей на монтажной плате
	\Bottomlayer_PIC_Eval.pdf	Чертеж печатной платы, вид снизу

(продолжение)

Папка	Файл	Описание
	\\Schaltplan_PIC_Eval.pdf	Принципиальная схема монтажной платы
	\\Stückliste_PIC-Eval.pdf	Спецификация монтажной платы (нем.)
	\\Toplayer_PIC_Eval.pdf	Чертеж печатной платы, вид сверху (со стороны деталей)
\\Entwicklungsboard \\Datenblaetter	\\BC547_Fairchild.pdf	Технический паспорт на транзистор BC547 компании Semiconductor (англ.)
	\\Display_EA_dog-m.pdf	Технический паспорт на ЖКИ EA DOGM162 (нем.)
	\\EEPROM_24LC32A_Microchip.pdf	Технический паспорт на EEPROM-память 24LC32A производства компании Microchip (англ.)
	\\IR-Empfänger-Modul_TSOP_1736_Vishay.pdf	Технический паспорт на микросхему инфракрасного приемника TSOP1736 производства компании Vishay (англ.)
	\\L7805_ST_Spannungsregler.pdf	Технический паспорт на стабилизатор L7805 (англ.)
	\\LM7805_Fairchild.pdf	Технический паспорт на стабилизатор LM7805 (англ.)
	\\PIC16F876A_DataSheet.pdf	Технические характеристики на микроконтроллер PIC16F876A (англ.)
	\\PIC16F87XA_Errata.pdf	Замечания опечатки в документации по семейству микроконтроллеров PIC16F87XA (англ.)
	\\RS232_Converter_MAX220-MAX249.pdf	Техническое описание преобразователей MAX220-MAX249 для преобразования сигналов ТТЛ/КМОП-уровня в сигналы интерфейса RS-232 (англ.)
	\\ST7036_LCD-Controller_EADOG.pdf	Техническое описание контроллера ST7036 для ЖКИ (англ.)
\\Entwicklungsboard \\Layoutdaten_Eagle		Рекомендации по установке ПО Eagle
\\Mplab		Документация и ПО для среды программирования MPLAB
\\Mplab\\Dokumente		Документация для среды программирования MPLAB
	\\MPASM_Assembler_UserGuide.pdf	Руководство пользователя MPASM (англ.)
	\\MPLAB-IDE_UserGuide.pdf	Руководство пользователя MPLAB-IDE (англ.)

(окончание)

Папка	Файл	Описание
\Mplab\Installation		Дистрибутив для установки ПО MPLAB v810
\Mplab\ReleaseNotes		Обзор ПО MPLAB
\Programmierung		Документы и ПО для программирования микроконтроллера
\Programmierung\MPLAB_ICD2		Документация по программированию в среде MPLAB_ICD2
	\FlashMemoryProgramming_Spec_PIC16F876A.pdf	Описание программирования Flash-памяти (англ.)
	\MPLAB_ICD2_In-Circuit-Debugger_UserGuide.pdf	Руководство пользователя для встроенного отладчика MPLAB_ICD2 (англ.)
\Programmierung\Selbstbau\Programmiertool		Документация и ПО для самостоятельного изготовления программатора
	\Kurzanleitung_WinPic_DE.pdf	Краткая инструкция по работе с программатором WinPic (нем.)
	\Kurzanleitung_WinPic_ru.pdf	Краткая инструкция по работе с программатором WinPic (рус.)
\Programmierung\Selbstbau\Programmiertool\WinPic		Дистрибутив для установки ПО WinPic
\Programmierung\Selbstbau\Schaltplan	\Mini_Pic_Programmer.pdf	Принципиальная схема программатора Mini_Pic

Предметный указатель

A

Acknowledge 238
ADC (Analog-to-Digital Converter) 149
ALU 6
ARM9 6
ASCII 28
ASCII-код 28
Assembler 1

B

BGA 6
Breakpoints 63
BRG 280
Brown-Out Detect 102

C

CCP 141
CCP1 124, 264, 266
CCP2 265
CISC 21
CISC-процессор 21
Code Protection 104
Compiler 9
COM-порт 214

D

DAC (Digital-to-Analog Converter) 149
DDRAM 170

E

EEPROM 1, 16, 112, 229
Electronic Assembly 163

F

File Register 9
Flash Program Memory 8
Flash-память 8

G

GPIO 121
GPR 11

H

HyperTerminal 221

I

PC 227, 272
PC-шина 112
ICD 5
ICD2 1, 85
Interrupt 51
IR-Protocol 245

L

Label 25
LED 126
LIFO 12
Linker 9
Low Voltage Program 103
LSB 38, 147

M

Master 227
Microchip 5
MPLAB 1, 63
MSB 38
MSSP 229, 232, 264

O

Open Collector 228
OTP 105

P

PC (Program Counter) 9, 47
PIC 5
PIC10F222 6

PIC16F876 6
 PIC16F876A 5
 ◊ блок-схема 6
 PIC24FJ128 6
 PIC32MX460 6
 Power Up Timer 102
 Pull-Down Resistor 96
 PWM 141

R

RAM 8
 RC5-код 245
 RISC 21
 RISC-процессор 21

S

SFR 11
 Sitronix 164

A

Адрес устройства 227
 АЛУ 6, 9
 Аналоговый вход 127
 Аналоговый сигнал 145
 Аппаратные средства 107
 Асинхронная передача данных 216
 Ассемблер 1, 8
 АЦП (аналого-цифровой преобразователь) 4,
 149, 179
 ◊ разрешающая способность 146

Б

Банк оперативной памяти 11
 Безусловный переход 47
 Биты конфигурации 88, 99, 104, 116
 Биты состояния 10
 Блок-схема 6

В

Ведомое устройство 227
 Ведущее устройство 227
 Вершина стека 12
 Вложенная подпрограмма 13
 Внутрисхемный отладчик 85
 Вход 126

Slave 227
 SLEEP 61, 100
 SMBus 272
 SPI 166, 271

T

TWI 227

U

USART 278, 280

W

Watchdog-Timer 101

Выбор банка памяти 11
 Выход 126
 Выходной делитель частоты 141
 Выходы с открытым коллектором 228
 Вычитание 16-битное 157

Д

Двоичное деление 196
 Двоичное умножение 192
 Дизассемблер 76
 Директива:
 ◊ #DEFINE 116
 ◊ #INCLUDE 116
 ◊ _CONFIG 116
 ◊ END 119
 ◊ ENDM 117
 ◊ EQU 117
 ◊ LIST 115
 ◊ MACRO 117
 ◊ ORG 118
 Дисплей 114, 163
 Дистанционное управление 245
 Дребезг контактов 225

Ж

ЖКИ (жидкокристаллический индикатор) 114,
 163, 164

3

Защита:

- ◊ кода 104
- ◊ чтения 104
- ◊ чтения данных из EEPROM-памяти 103

И

Инициализация 119

Интегрированная среда проектирования 63

Интерфейс:

- ◊ I²C 112
- ◊ RS-232 113, 213, 214
- ◊ SPI 122
- ◊ USB 213
- ◊ последовательный 213
- ◊ программирования 108
- Инфракрасное дистанционное управление 112

К

Кварц 100

- ◊ часовой 101

Кварцевый резонатор 100

Код:

- ◊ ASCII 28
- ◊ RC5 245
- ◊ машинный 9

Команда:

- ◊ addlw 44
- ◊ addwf 43
- ◊ andlw 31
- ◊ andwf 30
- ◊ bcf 37
- ◊ bsf 38
- ◊ btfsc 57
- ◊ btfss 55
- ◊ call 48
- ◊ clrf 35
- ◊ clrw 36
- ◊ clrwdt 59
- ◊ comf 36
- ◊ decf 46
- ◊ decfsz 54
- ◊ goto 47
- ◊ incf 45
- ◊ incfsz 52
- ◊ iorlw 32
- ◊ iorwf 32
- ◊ movf 42

- ◊ movlw 41
- ◊ movwf 41
- ◊ nop 53, 58
- ◊ retfie 51
- ◊ retlw 50
- ◊ return 49
- ◊ rlf 38
- ◊ rrf 40
- ◊ sleep 60
- ◊ sublw 45
- ◊ subwf 44
- ◊ swapf 42
- ◊ xorlw 34
- ◊ xorwf 33
- Комплятор 9
- Компоновщик 9
- Контроллер индикатора 164
- Контрольные точки останова 63
- Косвенная адресация 14

Л

Логический анализатор 82

М

Макрокоманда 11, 21, 117, 139, 174

Макросы 117

Маскирование 31

Метка 25

МЗР 147

Микрокомпьютер 3

Микроконтроллер 3

- ◊ PIC10F222 6

- ◊ PIC16F876 6

- ◊ PIC16F876A 1, 5

- ◊ PIC24FJ128 6

- ◊ PIC32MX460 6

- ◊ тп 66

Младший разряд двончного числа 38

Монтажная плата 107

Мощность 191

Н

Наблюдение 75

Набор символов 164

Напряжение:

- ◊ питания 108
- ◊ программирования 97
- Низковольтное программирование 103

О

Обозначения выводов 123
 ОЗУ 8
 Операционное устройство 6, 9
 Опорное напряжение 128, 146

П

Память EEPROM 1, 16, 112, 229
 Персональный компьютер (ПК) 213
 Пиксел 164
 Подпрограмма 12
 ◊ вложенная 13
 Подтягивающие резисторы 228
 Последовательный интерфейс 213
 Предварительный делитель частоты 134, 141
 Предделитель частоты 134
 Прерывание 51
 Приемник инфракрасного излучения 112
 Провал напряжения 102
 Программа для работы на терминале 217
 Программатор 84, 91
 Программирование:
 ◊ микроконтроллера 91
 ◊ низковольтное 103
 Программные настройки 88
 Программный интерфейс 95
 Программный счетчик 9, 47
 Проект 65
 Протокол инфракрасный (ИК) 245
 Процессор:
 ◊ CISC 21
 ◊ RISC 21

Р

Рабочая среда 64
 Рабочий стол 69
 Разъем расширения 114
 Расположение выводов 122
 Регистр:
 ◊ косвенной адресации 15
 ◊ переносов 141
 ◊ состояния 10
 ◊ общего назначения 11
 ◊ специального назначения 11
 ◊ режим ожидания 61, 100

С

Сброс 118
 Симулятор 72, 78

Скорость передачи информации 219
 Сложение 16-битное 156
 Согласующий резистор 96
 Сопротивление 191
 Старший разряд двончного числа 38
 Стек 12
 Стимул 78
 ◊ асинхронный 79
 Счетчик команд 9

Т

Таймер 133
 ◊ включения питания 102
 Текстовый редактор 92
 Телеуправление 112
 Тип генератора 100
 Тип микроконтроллера 66
 Точки останова 78

У

Указатель 15
 Управляющие символы 224

Ф

Флаг:
 ◊ десятичного переноса 10
 ◊ нулевого результата 10
 ◊ переноса 10
 ◊ состояния 10
 Флэш-память программ 8

Ц

ЦАП (цифроаналоговый преобразователь) 149

Ч

Чертеж размещения деталей 107

Ш

ШИМ (Широтно-импульсная модуляция) 141
 Шина PC 122

Э

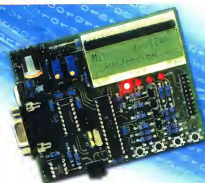
Электрическая схема монтажной платы 107

Я

Ядро ARM 6

Электроника

МИКРОКОНТРОЛЛЕРЫ для начинающих



В книге рассмотрены:

- Структура и принцип работы PIC16F876A
- Программирование с помощью MPLAB
- Обработка аналоговых сигналов
- Отображение данных на дисплее
- Измерение напряжения и мощности
- Передача данных через последовательный интерфейс
- Обмен данными с компьютером и многое другое

Сегодня микроконтроллеры управляют большинством современных электронных устройств. Между тем запрограммировать и использовать микроконтроллер для своих целей не так уж сложно. О том, как это сделать, рассказано в данной книге.

На практических примерах использования популярного микроконтроллера PIC16F876A компании Microchip вы научитесь: отображать данные на дисплее, измерять и записывать в память аналоговые сигналы, управлять выходами микроконтроллера с помощью ИК-пульта дистанционного управления и многое другое. Для программирования микроконтроллера используется внутрисхемный отладчик-программатор ICD 2, кроме того, приведена простая схема для программирования PIC-микроконтроллера через последовательный интерфейс. Подробно описаны основные команды языка ассемблер, а также среда разработки MPLAB. Для проведения экспериментов и изучения примеров приведено описание небольшой монтажной платы, которую можно приобрести на сайте автора (www.edmh.de) или изготовить самостоятельно, используя чертеж из книги.



БХВ-ПЕТЕРБУРГ
190005, Санкт-Петербург,
Измайловский пр., 29
E-mail: mail@bhv.ru
Internet: www.bhv.ru
Тел.: (812) 251-42-44
Факс: (812) 320-01-79



FRANZIS

И. КОДМАНДИН

МЫЖЕДОУПРОВА
ПЕРВАЯ

ПРАВИЛА

